



**Bruno Miguel Nunes da Silva**

MSc

# **Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps**

Dissertação para obtenção do Grau de Doutor em  
Informática

Orientadores : Doutor Nuno Cavalheiro Marques, Prof. Auxiliar,  
Universidade Nova de Lisboa  
Doutor Octavian Postolache, Prof. Auxiliar,  
Instituto Universitário de Lisboa

Júri:

Presidente: Doutor José Júlio Alves Alferes

Arguentes: Doutor Victor José de Almeida e Sousa Lobo  
Doutor João Manuel Portela da Gama

Vogais: Doutor Paulo Alexandre Ribeiro Cortez  
Doutora Carmen Pires Morgado



FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE NOVA DE LISBOA

**Dezembro, 2016**



## **Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps**

Copyright © Bruno Miguel Nunes da Silva, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objectivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

---

A elaboração desta tese foi co-financiada pela bolsa PROTEC SFRH/BD/49723/2009 da Fundação para a Ciência e Tecnologia e beneficiou do regime de isenção de propinas de doutoramento, no âmbito do Protocolo de Cooperação existente entre a Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa e o Instituto Politécnico de Setúbal.





*À minha família*



# Acknowledgements

Science is organized knowledge. Wisdom is organized life.

---

IMMANUEL KANT, GERMAN PHILOSOPHER (1724-1804)

This work could not have been accomplished without the help and support of several people and institutions, to which I hereby thank.

First of all, I thank *Faculdade de Ciências e Tecnologia — Universidade Nova de Lisboa*, for allowing me to endure the doctoral program.

This thesis was jointly supervised by Prof. Nuno Marques, Assistant Professor at *Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa* and Prof. Octavian Postolache, Assistant Professor at *ISCTE — Instituto Universitário de Lisboa*, to whom I am grateful for accepting this supervision and for their invaluable support, mentoring, guidance and collaboration.

My acknowledgements extend to the remaining members of the *Thesis Advisory Committee*, for their follow-up and suggestions.

This work was partially funded by *Fundação para a Ciência e a Tecnologia*, with a PhD grant (SFRH/BD/49723/2009), through the PROTEC program. It also benefited from tuition exemption, in agreement with the cooperation protocol established between *Faculdade de Ciências e Tecnologia — Universidade Nova de Lisboa* and *Instituto Politécnico de Setúbal*.

I would also like to thank *GoBusiness Finance* and *Agência Portuguesa do Ambiente* for providing data that was valuable for this thesis.

Most of all, an emotional THANK YOU to my family for all their unconditional support and for making me the man that I am today. Without them, none of this would have been possible.



# Abstract

---

This thesis addresses the use of Self-Organizing Maps (SOM) for exploratory cluster analysis over ubiquitous data streams, where two complementary problems arise: first, to generate (local) SOM models over potentially unbounded multi-dimensional non-stationary data streams; second, to extrapolate these capabilities to ubiquitous environments. Towards this problematic, original contributions are made in terms of algorithms and methodologies. Two different methods are proposed regarding the first problem. By focusing on visual knowledge discovery, these methods fill an existing gap in the panorama of current methods for cluster analysis over data streams. Moreover, the original SOM capabilities in performing both clustering of observations and features are transposed to data streams, characterizing these contributions as versatile compared to existing methods, which target an individual clustering problem. Also, additional methodologies that tackle the ubiquitous aspect of data streams are proposed in respect to the second problem, allowing distributed and collaborative learning strategies.

Experimental evaluations attest the effectiveness of the proposed methods and real-world applications are exemplified, namely regarding electric consumption data, air quality monitoring networks and financial data, motivating their practical use.

This research study is the first to clearly address the use of the SOM towards ubiquitous data streams and opens several other research opportunities in the future.

**Keywords:** self-organizing maps; data streams; ubiquitous data mining; cluster analysis; exploratory knowledge discovery.

---



# Resumo

---

Esta tese aborda o uso de mapas auto-organizados (SOM) para *clustering* exploratório de dados sobre *streams* de dados ubíquas, onde dois problemas complementares surgem: primeiro, gerar modelos SOM (locais) sobre *streams* de dados não estacionárias e potencialmente infinitas; segundo, extrapolar estas capacidades para ambientes ubíquos. Contribuições originais são apresentadas em termos de algoritmos e metodologias na solução deste problema. Dois métodos diferentes são propostos para o primeiro problema. Focados na descoberta de conhecimento visual, estes métodos preenchem uma lacuna no panorama de métodos atuais para *clustering* em *streams* de dados. Adicionalmente, as capacidades originais do SOM relativamente a *clustering* de observações e de variáveis são transpostas para *streams* de dados, caracterizando estas contribuições como versáteis comparativamente aos métodos existentes, que focam isoladamente um dos tipos de *clustering*. Adicionalmente, metodologias que abordam o aspeto distribuído das *streams* de dados são apresentadas para o segundo problema, permitindo estratégias de aprendizagem distribuídas e colaborativas.

Avaliações experimentais atestam a efetividade dos métodos para os problemas propostos e aplicações no mundo real são exemplificadas, nomeadamente com dados de consumo elétrico, de redes de monitorização de qualidade do ar e financeiros, motivando o seu uso prático.

Este estudo é o primeiro a abordar de forma clara o uso do SOM relativamente a *streams* de dados ubíquas e apresenta inúmeras oportunidades futuras de investigação.

**Palavras-chave:** mapas auto-organizados; *streams* de dados; data mining em ambientes ubíquos; *clustering* de dados; descoberta de conhecimento exploratória

---





# Contents

<b>Contents</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xix</b>
<b>List of Tables</b>	<b>xxiii</b>
<b>List of Abbreviations</b>	<b>xxv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Motivation . . . . .	3
1.3 Contributions . . . . .	5
1.3.1 Publications . . . . .	7
1.4 Personal Note . . . . .	7
1.5 Thesis Structure . . . . .	8
1.5.1 Basis and Aim (Chapters 1, 2 and 3) . . . . .	9
1.5.2 Methods, Results and Discussion (Chapters 4, 5, 6 and 7) . . . . .	9
1.5.3 Final Remarks (Chapter 8) . . . . .	9
1.5.4 Appendices . . . . .	9
<b>2 Rationale and Literature Review</b>	<b>11</b>
2.1 Chapter Overview . . . . .	11
2.2 Motivation and Aim . . . . .	12
2.3 Unsupervised Knowledge Discovery from Data Streams . . . . .	12
2.3.1 The Data Stream Model . . . . .	13
2.3.2 Motivating Scenarios and Applications . . . . .	15
2.4 The Self-Organizing Map . . . . .	17
2.4.1 Competitive Learning Fundamentals . . . . .	17
2.4.2 SOM Model and Algorithm . . . . .	21
2.4.3 Related Algorithms . . . . .	23
2.4.4 Ordering, Convergence and Parameterization . . . . .	25

2.4.5	Visualizations and Exploratory Knowledge Discovery . . . . .	29
2.4.6	Quality Assessment . . . . .	32
2.4.7	Comparison of Clustering Methods . . . . .	34
2.5	Evaluation of the SOM and Related Variants for Data Streams . . . . .	42
2.5.1	Proposed Requirements . . . . .	43
2.5.2	Variants of the SOM . . . . .	43
2.5.3	Evaluation . . . . .	48
2.6	Current Methods for Cluster Analysis on Data Streams . . . . .	50
2.6.1	Clustering Observations . . . . .	50
2.6.2	Clustering Features . . . . .	55
2.6.3	Collaborative and Distributed Learning . . . . .	56
2.7	Critical Overview . . . . .	57
2.7.1	Research Paths and Additional Research Questions . . . . .	61
<b>3</b>	<b>Aims and Methodology Overview</b>	<b>63</b>
3.1	A Two-Phase ART/SOM Approach to Data Streams . . . . .	64
3.2	The Ubiquitous Self-Organizing Map for Non-Stationary Data Streams . .	65
3.3	Distributed and Collaborative Learning of Ubiquitous Data Streams . . .	66
3.4	Real-World Applications . . . . .	67
3.5	Considerations on the Normalization of Data Streams . . . . .	67
3.6	Artificial Data Streams . . . . .	69
<b>4</b>	<b>A Two-Phase ART/SOM Approach to Data Streams</b>	<b>71</b>
4.1	Chapter Overview . . . . .	71
4.2	Motivation and Aim . . . . .	72
4.3	A Two-Phase StreamART2A/SOM Methodology . . . . .	72
4.3.1	Methodology Overview . . . . .	73
4.3.2	Notation . . . . .	74
4.3.3	The StreamART2A Algorithm . . . . .	74
4.3.4	Modified Batch SOM Algorithm . . . . .	78
4.3.5	Trend of Model Adaptation by the Average Quantization Error . .	79
4.4	StreamART2A Algorithm Analysis . . . . .	79
4.4.1	Time and Space Complexity . . . . .	81
4.5	Experimental Evaluation . . . . .	82
4.5.1	Overview . . . . .	83
4.5.2	Data Normalization and Algorithm Parameters . . . . .	84
4.5.3	Parameter Sensitivity Analysis . . . . .	84
4.5.4	Vigilance and Landmark Window Illustrations . . . . .	88
4.5.5	Exploratory Cluster Analysis . . . . .	93
4.5.6	Model Assessment . . . . .	95
4.6	Remarks and Future Work . . . . .	97

<b>5</b>	<b>The Ubiquitous Self-Organizing Map for Non-Stationary Data Streams</b>	<b>101</b>
5.1	Chapter Overview . . . . .	101
5.2	Motivation and Aim . . . . .	102
5.3	The Ubiquitous Self-Organizing Map . . . . .	103
5.3.1	Algorithm Overview . . . . .	103
5.3.2	Notation . . . . .	104
5.3.3	Online Assessment Metrics . . . . .	106
5.3.4	The Drift Function . . . . .	109
5.3.5	The Neighborhood Function . . . . .	110
5.3.6	States and Algorithm Formalization . . . . .	111
5.4	UbiSOM Algorithm Analysis . . . . .	114
5.4.1	Time and Space Complexity . . . . .	117
5.5	UbiSOM Experimental Evaluation . . . . .	117
5.5.1	Experimental Evaluation Overview . . . . .	117
5.5.2	Data Normalization and Algorithm Parameters . . . . .	118
5.5.3	Parameter sensitivity analysis . . . . .	118
5.5.4	Convergence with stationary and non-stationary data . . . . .	123
5.5.5	Exploratory Cluster Analysis . . . . .	131
5.5.6	Comparison against other variants . . . . .	132
5.6	Feature Clustering Methodology and Evaluation . . . . .	141
5.6.1	Methodology . . . . .	141
5.6.2	Evaluation . . . . .	142
5.6.3	Data and Parameterization . . . . .	142
5.6.4	Results . . . . .	142
5.7	Assessment Regarding Established Requirements . . . . .	144
5.8	Remarks and Future Work . . . . .	147
<b>6</b>	<b>Distributed and Collaborative Learning of Ubiquitous Data Streams</b>	<b>151</b>
6.1	Chapter Overview . . . . .	151
6.2	Motivation and Aim . . . . .	152
6.3	Distributed and Collaborative Learning . . . . .	153
6.3.1	Methodology Overview . . . . .	153
6.3.2	Notation . . . . .	154
6.3.3	General Techniques . . . . .	156
6.3.4	Distributed Learning . . . . .	157
6.3.5	Collaborative Learning . . . . .	158
6.4	Experimental Evaluation . . . . .	160
6.4.1	Artificial Data . . . . .	160
6.5	Remarks and Future Work . . . . .	169

<b>7</b>	<b>Real-World Applications</b>	<b>173</b>
7.1	Chapter Overview . . . . .	173
7.2	Motivation and Aim . . . . .	174
7.3	Household Electric Power Consumption . . . . .	174
7.3.1	Data Description . . . . .	175
7.3.2	Parameterization . . . . .	177
7.3.3	The BMU Map and Activity Map Visualizations . . . . .	177
7.3.4	Application Results . . . . .	179
7.4	Distributed Air Quality Monitoring . . . . .	185
7.4.1	Data Description . . . . .	189
7.4.2	Parameterization . . . . .	191
7.4.3	Application Results . . . . .	191
7.5	Feature Clustering in Financial Data . . . . .	197
7.5.1	Data Description . . . . .	198
7.5.2	Parameterization . . . . .	199
7.5.3	Application Results . . . . .	201
7.6	Remarks and Future Work . . . . .	209
<b>8</b>	<b>Conclusions</b>	<b>213</b>
8.1	Main Findings and Contributions . . . . .	213
8.1.1	Summary of Contributions . . . . .	216
8.1.2	Requirements for SOM Variants in Dealing with Data Streams . . . . .	217
8.1.3	A Two-Phase ART/SOM Approach to Data Streams . . . . .	218
8.1.4	The Ubiquitous Self-Organizing Map . . . . .	219
8.1.5	Distributed and Collaborative Learning of Ubiquitous Data Streams . . . . .	220
8.1.6	Real-World Applications . . . . .	221
8.1.7	UbiSOM Framework and Library . . . . .	223
8.1.8	Overall Assessment . . . . .	223
8.2	Future Work and Open Issues . . . . .	225
8.3	Final Remark . . . . .	228
	<b>Bibliography</b>	<b>229</b>
<b>A</b>	<b>Contributed Framework and Software</b>	<b>239</b>
A.1	Framework . . . . .	239
A.1.1	Local applications . . . . .	241
A.1.2	Remote applications . . . . .	242
A.2	Library . . . . .	245
A.2.1	Interactive Demonstration . . . . .	245
A.2.2	Color Quantization in a Video Stream . . . . .	246

<b>B Artificial Data Streams</b>	<b>251</b>
B.1 Gauss . . . . .	251
B.2 Complex . . . . .	251
B.3 Chain . . . . .	252
B.4 d2k20, d3k20 and d5k20 . . . . .	252
B.5 AbruptOneTwo . . . . .	252
B.6 DriftTwoOne . . . . .	253
B.7 Clouds . . . . .	253
B.8 Hepta . . . . .	253
B.9 Correlated . . . . .	253
<b>C Additional Results from Experimental Evaluations</b>	<b>259</b>
C.1 StreamART2A Algorithm . . . . .	259
C.1.1 Parameter Sensitivity Analysis . . . . .	259
C.2 UbiSOM Algorithm . . . . .	264
C.2.1 Description and Algorithm Parameters . . . . .	264
C.2.2 Parameter Sensitivity Analysis . . . . .	264
C.2.3 Convergence with stationary and non-stationary data . . . . .	265
C.2.4 Exploratory Cluster Analysis . . . . .	270
<b>D Triple-Cascaded Moving Average</b>	<b>277</b>



# List of Figures

1.1	Problem statement illustration. . . . .	2
1.2	Thesis structure. . . . .	8
2.1	Architecture of the competitive learning network. . . . .	18
2.2	Structure of a $8 \times 5$ SOM network. . . . .	21
2.3	Ordering and convergence phases of the SOM over an uniform 2D distribution. . . . .	26
2.4	Topological ordering of the SOM. . . . .	28
2.5	Depiction of a $20 \times 40$ SOM quantizing the IRIS dataset. . . . .	31
2.6	Examples of visualizations derived from a $20 \times 40$ SOM after learning the Iris dataset. . . . .	32
2.7	Comparison of clustering results by different algorithms. . . . .	36
2.8	Two-stage clustering of the SOM. . . . .	40
2.9	Iris dataset represented as a data stream. . . . .	42
2.10	Panorama of available methods for clustering data streams before research. . . . .	60
3.1	Maximum distance in the assumed normalized input space of data streams. . . . .	68
4.1	StreamART2A/SOM methodology overview. . . . .	73
4.2	StreamART2A: example of procedures in a landmark window, where $q$ equal to the number of underlying clusters. . . . .	80
4.3	StreamART2A: alternate result regarding Figure 4.2, with $q$ greater than the number of underlying clusters. . . . .	81
4.4	A StreamART2A codebook implemented by means of a balanced binary tree. . . . .	82
4.5	Parameter sensitivity analysis results for the StreamART2A algorithm over stationary data streams. . . . .	85
4.6	Parameter sensitivity analysis results for the StreamART2A algorithm over non-stationary data streams. . . . .	86
4.7	Impact of landmark window size $L$ in mean number of merges. . . . .	88
4.8	Impact of data dimensionality in mean number of merges. . . . .	88

4.9	Impact of data dimensionality in the computational cost of a landmark window. . . . .	89
4.10	Generated micro-categories for first landmark window over the <i>Complex</i> data stream. . . . .	91
4.11	Generated micro-categories for first landmark window over the <i>Chain</i> data stream. . . . .	91
4.12	Generated micro-categories for first landmark window over the <i>d3k20</i> data stream. . . . .	92
4.13	Obtained <i>d5k20</i> offline SOM models for $t_1 = 0, t_2 = 20\,000$ , with varying $q$ values. . . . .	93
4.14	Batch SOM models, and derived U-Matrices, generated from specific time intervals over the stationary <i>Gauss</i> , <i>Complex</i> and <i>Chain</i> data streams. . . .	94
4.15	U-Matrices derived from Batch SOM models generated from specific time intervals over the non-stationary <i>Clouds</i> and <i>Hepta</i> data streams. . . . .	96
4.16	Behavior of $E'_q(t)$ and $\overline{qe}(t)$ over various data streams. . . . .	98
5.1	Example of behavior of local $E'_q(t)$ vs. average quantization error $\overline{qe}(t)$ . . .	107
5.2	Example of a distribution change not detected by the <i>average quantization error</i> . . . . .	108
5.3	Behavior of a simple moving average <i>versus</i> a triple cascaded moving average in the presence of abrupt change. . . . .	110
5.4	Maximum lattice distance $\parallel \text{DIAG} \parallel$ used for normalization of $\sigma(t)$ . . . .	112
5.5	UbiSOM neighborhood kernel $h'_{ck}$ for different values of $\sigma$ . . . . .	112
5.6	UbiSOM states and transitions. . . . .	112
5.7	The UbiSOM algorithm seen as a feedback control system. . . . .	114
5.8	The UbiSOM evolution over the stationary <i>Complex</i> data stream. . . . .	124
5.9	The UbiSOM evolution over the stationary <i>Chain</i> data stream. . . . .	125
5.10	The UbiSOM evolution over the stationary <i>Clouds</i> data stream. . . . .	126
5.11	The UbiSOM evolution over the stationary <i>Hepta</i> data stream. . . . .	127
5.12	U-Matrices obtained for stationary data streams. . . . .	131
5.13	U-Matrices obtained for non-stationary data streams. . . . .	133
5.14	Final maps of different SOM variants obtained for the stationary Gaussian data stream. . . . .	134
5.15	Average quantization error $\overline{qe}(t)$ of tested SOM variants across all data streams. . . . .	136
5.16	Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PLSOM at the end of the stationary <i>Chain</i> data stream. . . . .	137
5.17	Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PLSOM at the end of the non-stationary <i>Clouds</i> data stream. . . . .	138
5.18	Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PLSOM at the end of the non-stationary <i>Hepta</i> data stream. . . . .	139



5.19	Learning evolution of UbiSOM over the <i>Correlated</i> data stream. . . . .	143
5.20	Feature clustering from UbiSOM model at $t = 30\,000$ when learning the <i>Correlated</i> data stream. . . . .	145
5.21	Feature clustering from UbiSOM model at $t = 80\,000$ when learning the <i>Correlated</i> data stream. . . . .	146
6.1	Motivating scenario for distributed and/or collaborative learning strategies.	152
6.2	The merge procedure aims at removing duplicate prototypes from a pair of codebooks $\mathcal{A}$ and $\mathcal{B}$ that roughly overlap in the input space. . . . .	157
6.3	Distributed learning from distributed local UbiSOM models. . . . .	158
6.4	Collaborative learning framework. . . . .	159
6.5	Generation of distributed data from the <i>Complex</i> artificial data stream. . .	161
6.6	Sequence of events in the simulated ubiquitous environment. . . . .	162
6.7	Final obtained local models in the collaborative learning methodology for the <i>Complex</i> distribution. . . . .	166
6.8	Final obtained global models in the collaborative learning methodology for the <i>Complex</i> distribution. . . . .	167
6.9	Filtered vs unfiltered centralized codebooks from individual nodes. . . .	169
6.10	Centralized model and U-Mat. . . . .	170
7.1	Household data stream in the year 2007. . . . .	178
7.2	Assessment metrics and learning parameters evolution in the <i>Household</i> data stream. . . . .	179
7.3	Household Winter model and additional proposed visualizations. . . . .	181
7.4	Preceding 48h of observations for the Household Winter model. . . . .	182
7.5	Household Autumn model and additional proposed visualizations. . . .	183
7.6	Preceding 48h of observations for the Household Autumn model. . . . .	184
7.7	Location and characterization of air quality monitoring stations in Portugal.	187
7.8	Air quality monitoring stations in the district of Aveiro. . . . .	189
7.9	Obtained complete cases for the QualAr data streams. . . . .	191
7.10	QualAr data streams after pairwise deletion of missing values. . . . .	192
7.11	<i>QualAr Summer</i> models — derived U-Matrices and component planes. . .	194
7.12	<i>QualAr Winter</i> models — derived U-Matrices and component planes. . .	195
7.13	Mapping of AQI values against $PM_{10}$ denormalized component planes of both centralized <i>QualAr Summer</i> and <i>Winter</i> models. . . . .	196
7.14	<i>Financial data stream</i> . . . . .	200
7.15	Component planes of Financial Snapshot 1. . . . .	202
7.16	Component planes of Financial Snapshot 2. . . . .	203
7.17	Component planes of Financial Snapshot 3. . . . .	204
7.18	Feature clustering results for the SOM models obtained along the <i>Financial data stream</i> . . . . .	206

7.19	Entanglement between consecutive financial clustering results. . . . .	208
7.20	Average quantization errors for the Financial data stream. . . . .	209
8.1	Panorama of available methods for clustering data streams after research. . . . .	224
A.1	Framework as a layered architecture. . . . .	240
A.2	Configuration file for the developed framework. . . . .	241
A.3	Local application that allows real-time visualization of the UbiSOM model. . . . .	242
A.4	Functionalities of the remote exploratory cluster analysis application. . . . .	243
A.5	Remote visualization through Android application. . . . .	244
A.6	R Studio integration through developed R library. . . . .	244
A.7	UbiSOM library class diagram. . . . .	245
A.8	UbiSOM Interactive Demo application. . . . .	246
A.9	Evolution of UbiSOM learning parameters over the SINTEL video frames. . . . .	247
A.10	SINTEL video frames and UbiSOM codebooks. . . . .	248
A.11	Quantized SINTEL video frames by the UbiSOM algorithm. . . . .	249
B.1	Illustration of several stationary artificial data streams ( <i>Gauss</i> , <i>Complex</i> , <i>Chain</i> and <i>d3k20</i> ). . . . .	254
B.2	Illustration of non-stationary artificial data streams ( <i>AbruptOneTwo</i> and <i>DriftTwoOne</i> ). . . . .	255
B.3	Illustration of additional non-stationary artificial data streams ( <i>Clouds</i> and <i>Hepta</i> ). . . . .	256
B.4	Illustration of the non-stationary <i>Correlated</i> data stream. . . . .	257
C.1	Additional parameter sensitivity analysis results for the StreamART2A algorithm over <i>Gauss</i> and <i>Chain</i> stationary data streams. . . . .	261
C.2	Additional parameter sensitivity analysis results for the StreamART2A algorithm over <i>d2k20</i> and <i>d3k20</i> stationary data streams. . . . .	262
C.3	Additional parameter sensitivity analysis results for the StreamART2A algorithm over the non-stationary <i>Hepta</i> data stream. . . . .	263
C.4	The UbiSOM evolution over the stationary <i>Gauss</i> data stream. . . . .	271
C.5	The UbiSOM evolution over the stationary <i>AbruptOneTwo</i> data stream. . . . .	272
C.6	The UbiSOM evolution over the non-stationary <i>DriftTwoOne</i> data stream. . . . .	273
C.7	The UbiSOM evolution over the non-stationary <i>DriftTwoOne</i> data stream. . . . .	274
C.8	The UbiSOM evolution over the stationary <i>d5k20</i> data stream, final lattice and corresponding U-Matrix. . . . .	275
D.1	Triple moving average (3MA) filter, by cascading three moving averages of decreasing window sizes. . . . .	277
D.2	Comparison between MA and 3MA filters. . . . .	279

# List of Tables

2.1	Evaluation of SOM variants under proposed requirements. . . . .	49
2.2	Overview of state-of-the-art algorithms for cluster analysis over data streams. . . . .	52
3.1	Summary of artificial data streams. . . . .	70
4.1	Notation for the StreamART2A algorithm. . . . .	75
4.2	Experimental evaluation overview for the StreamART2A/SOM methodology. . . . .	83
4.3	Obtained vigilance values across all tested data streams for various parameterizations. . . . .	90
5.1	Notation for the UbiSOM algorithm. . . . .	105
5.2	Experimental evaluation overview for the UbiSOM algorithm. . . . .	118
5.3	Summary of the UbiSOM parameter sensitivity analysis. . . . .	121
5.4	Comparison of the UbiSOM, Online SOM and PLSOM algorithms across all data streams. . . . .	135
5.5	Underlying correlations in the artificial <i>Correlated</i> data stream. . . . .	143
6.1	Notation used in distributed and collaborative learning methodologies. . . . .	155
6.2	Parameters used in the simulated ubiquitous environment. . . . .	163
6.3	Sequence of collaborating events in the simulated ubiquitous environment. . . . .	165
6.4	Normalized mean quantization error of each global codebook $\mathcal{G}_i$ against the entire <i>Complex</i> data stream. . . . .	168
7.1	Data summary for UCI's "Individual household electric power consumption Data Set". . . . .	176
7.2	Optimization results for a PSA over the <i>Household</i> data stream (only with data from 2007). . . . .	178
7.3	Guideline values for the Air Quality Index (AQI) in Portugal. All values are in $\mu g/m^3$ . . . . .	188

7.4	Data summary of <i>QualAr</i> data obtained for Aveiro district air quality monitoring stations. . . . .	190
7.5	Optimization results for PSA analysis over <i>QualAr</i> data streams (only observations from 2014). . . . .	193
7.6	List of financial institutions (stock prices) contained in the <i>Financial data stream</i> . . . . .	199
B.1	Summary of artificial data streams. . . . .	252
B.2	Pairwise correlations in the <i>Correlated</i> artificial data stream. . . . .	254
C.1	Summary of additional experimental results for the StreamART2A algorithm. . . . .	260
C.2	Summary of additional experimental results for the UbiSOM algorithm. . . . .	264
C.3	UbiSOM parameter sensitivity analysis for <i>Gauss</i> data stream. . . . .	266
C.4	UbiSOM parameter sensitivity analysis for <i>Complex</i> data stream. . . . .	266
C.5	UbiSOM parameter sensitivity analysis for <i>Chain</i> data stream. . . . .	267
C.6	UbiSOM parameter sensitivity analysis for <i>d5k20</i> data stream. . . . .	267
C.7	UbiSOM parameter sensitivity analysis for <i>AbruptOneTwo</i> data stream. . . . .	268
C.8	UbiSOM parameter sensitivity analysis for <i>DriftTwoOne</i> data stream. . . . .	268
C.9	UbiSOM parameter sensitivity analysis for <i>Clouds</i> data stream. . . . .	269
C.10	UbiSOM parameter sensitivity analysis for <i>Hepta</i> data stream. . . . .	269

# List of Abbreviations

<b>3MA</b>	Triple Moving Average
<b>ANN</b>	Artificial Neural Network
<b>APA</b>	Agência Portuguesa do Ambiente
<b>AQI</b>	Air Quality Index
<b>ART</b>	Adaptive Resonance Theory
<b>BMU</b>	Best Matching Unit
<b>CCA</b>	Curvilinear Component Analysis
<b>CLIQUE</b>	Clustering In Quest
<b>DBSCAN</b>	Density-Based Spatial Clustering of Applications with Noise
<b>DSOM</b>	Dynamic Self-Organizing Map
<b>EDP</b>	Energias de Portugal
<b>EM</b>	Expectation-Maximization
<b>ESOINN</b>	Enhanced Self-Organizing Incremental Neural Network
<b>ESOM</b>	Evolving Self-Organizing Map
<b>GCS</b>	Growing Cell Structures
<b>GMM</b>	Gaussian Mixture Models
<b>GNG</b>	Growing Neural Gas
<b>IoT</b>	Internet of Things
<b>KDD</b>	Knowledge Discovery and Data Mining
<b>MA</b>	Moving Average

<b>MDS</b>	Multi-Dimensional Scaling
<b>MSE</b>	Mean Squared Error
<b>NG</b>	Neural Gas
<b>PCA</b>	Principal Component Analysis
<b>PDA</b>	Personal Digital Assistant
<b>PDF</b>	Probability Distribution Function
<b>PLSOM</b>	Parameter-less Self-Organizing Map
<b>PSA</b>	Parameter Sensitivity Analysis
<b>QE</b>	Quantization Error
<b>ROLF</b>	Regional and Online Learnable Fields
<b>SOINN</b>	Self-Organizing Incremental Neural Network
<b>SOM</b>	Self-Organizing Map
<b>TE</b>	Topographic Error
<b>UbiSOM</b>	Ubiquitous Self-Organizing Map
<b>UDM</b>	Ubiquitous Data Mining
<b>VP</b>	Vector Projection
<b>VQ</b>	Vector Quantization
<b>WTA</b>	Winner-take-all
<b>WTM</b>	Winner-take-most



# Introduction

An expert is a person who has made all the mistakes that can be made in a very narrow field.

---

NIELS BOHR, DANISH PHYSICIST (1885-1962)

## 1.1 Problem Statement

This thesis concerns the use of Self-Organizing Maps ([Kohonen 1982, 2001](#)) for exploratory cluster analysis over ubiquitous data streams. The *Self-Organizing Map* (SOM) is an unsupervised learning *artificial neural network* (ANN) model, based on *competitive learning*, that excels in data visualization. A standard SOM model consists in a set of topologically ordered data prototypes arranged in a rectangular lattice (the codebook), to where simple, but powerful, data visualizations are applied. It was initially proposed in 1982 and has become a well established data mining algorithm with hundreds of applications in many scientific domains ([Pöllä, Honkela, and Kohonen 2009](#)) — yet, in the context of *static data* and, therefore, assuming *stationary* distributions. However, *data streams* are characterized by unrestrained amounts of data being continuously generated and *non-stationarity*. Data streams are generated naturally within several applications as opposed to simple datasets, e.g., network monitoring, web mining, sensor networks, telecommunications, and financial applications. Moreover, in some of these, e.g., sensor networks, data streams may be generated in a distributed manner; also, with the advent of the *Internet of Things* (IoT) data gathering and processing have been increasingly disseminated in the physical space. This highlights the fact that data streams may also involve an *ubiquitous* aspect to them.

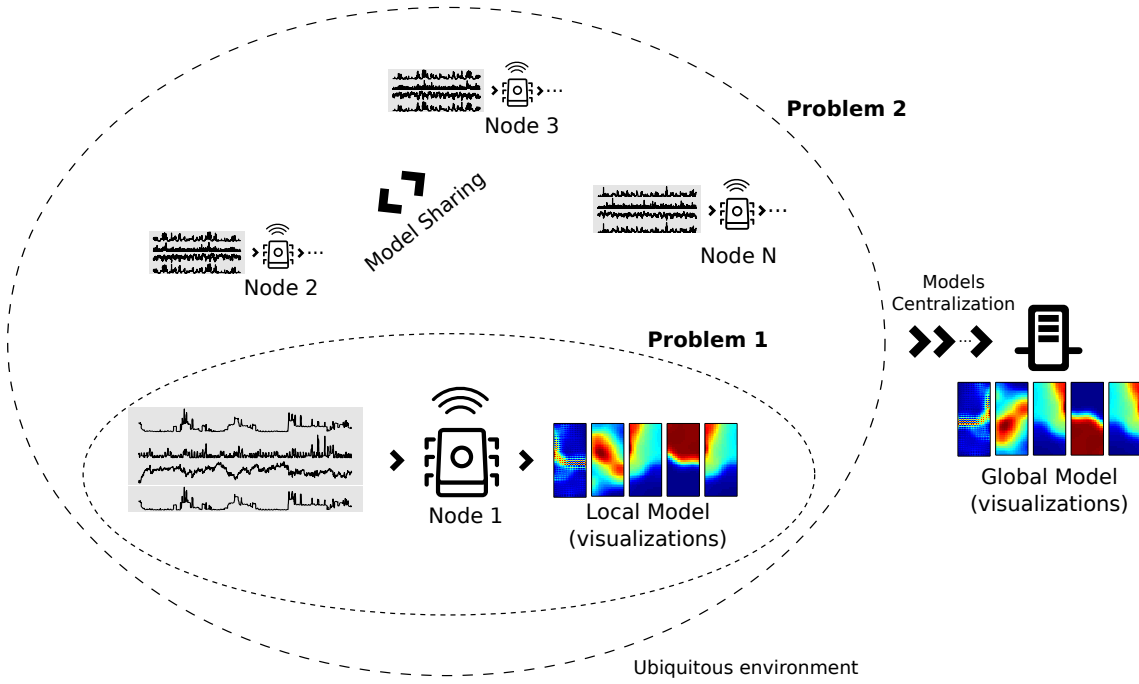


Figure 1.1: Problem statement illustration.

Consequently, thesis actually addresses two different, but complementary, problems as depicted in Figure 1.1:

**Problem 1** Generate SOM models over potentially unbounded multi-dimensional non-stationary data streams, where at any particular point in time a model can be obtained and exploratory cluster analysis performed (through visualization procedures) at each depicted node.

**Problem 2** Instead of centralizing the various data streams, which is not scalable, obtain global models through centralization of the local models themselves; also, if nodes are mobile, e.g., smartphones, to enable sharing of local models through opportunistic networks, so a node can “learn” from others.

These problems are dependent because the solution to the later problem inevitably depends on the former. Moreover, it should be noted that several application scenarios, i.e., monitoring applications, may require only the processing of a single multi-dimensional data stream. Therefore, the research effort was inevitably biased towards the former problem, which is sufficiently relevant by its own.

From the above problem statement the initial research questions and respective objectives towards their answers were thus the following:

**RQ1** *What are the limitations of Self-Organizing Maps regarding streaming data?*



**Objectives** Address the *data stream model* and challenges of cluster analysis over data streams; provide a deep understanding of the SOM algorithm and its strengths and weaknesses; identify what characteristics of the SOM algorithm prevents it from being applied to data streams; define requirements for SOM variants operating over data streams, and; review existing variants of the SOM that may target some (or all) of the previous defined requirements.

**RQ2** *Can Self-Organizing Maps provide a valuable tool for data stream cluster analysis regarding current methods? If so, which research paths should be pursued in terms of relevant contributions?*

**Objectives** Review current approaches to clustering data streams and establish the relevancy and innovative aspect of this research; identify proven methodologies from the previous review that may be explored in this study, and; define research paths to explore, given current state-of-the-art methodologies and limitations found in current SOM variants.

**RQ3** How to address the use of Self-Organizing Maps in ubiquitous environments?

**Objectives** Understand how the ubiquitous aspect of data streams is currently dealt with while providing a survey of such methodologies, and; suggest how the above methodologies can be addressed with the SOM and identify an additional research path regarding the ubiquitous aspect of data streams.

These research questions are addressed in detail in Chapter 2, where established research paths and additional research questions, that guided the contributions of this thesis, are formalized.

## 1.2 Motivation

Data streams have gained a lot of attention in academia in the recent past, given the sheer volume of data being produced continuously in a wide range of applications. Comparatively to static data, data streams impose harder challenges on algorithms operating over them, namely because they are potentially unbounded and non-stationary, i.e., the underlying distribution may change over time. The rationale behind data streams is that the volume and rate of data can be such that it is impractical to store them. Hence, clustering algorithms operating on data streams should ideally operate as single-pass algorithms and handle evolving data (Gama 2010) — see Section 2.3. Commonly agreed is the fact that such algorithms can only return approximate clustering results, since data cannot be revisited to fine-tune the models (Guha et al. 2003).

The SOM is widely used as a tool for projecting high-dimensional data onto a two-dimensional representation space (the map). This projection retains the relationship between input data as faithfully as possible, thus describing a topology-preserving representation of input similarities in terms of distances in the output space (see Section 2.4). The main advantage of such a projection is the ease by which a user can gain insight over the data structure by analyzing the map. Through visualization techniques readily applied to maps, it is possible to, e.g., visually identify clusters and cluster descriptions, without imposing any notion and number of clusters. Another motivating aspect for the use of SOMs regards two different clustering problems that can arise from data streams, namely the above traditional *clustering of observations* and the less traditional *clustering of features*, i.e., clustering individual time-series of a multi-dimensional data stream. While current proposals target these problems individually, the SOM algorithm is known for its versatility in performing both (Silva and Marques 2010a; Vesanto and Ahola 1999), but currently only in a static data context. Therefore, the extrapolation of these capabilities to a streaming setting is of great interest. However, the original SOM algorithms cannot be applied to data streams because they rely on annealing schemes to guide the learning parameters of the algorithm, i.e., data size must be known in advance. Yet, there have been some proposals of SOM variants that can deal with non-stationary data, however falling short on the exploratory cluster analysis capabilities (due to irregular network topologies) or not achieving a proper vector quantization mapping (which deters or degrades the application of the standard visualization techniques) — see Section 2.5.

Overall, the described characteristics and versatility of the SOM are not found in other current data stream clustering methods. Generally, current proposals involve the adaptation of traditional clustering algorithms to a streaming setting, where most apply a two-phase approach, i.e., maintaining *online* data abstractions from where clustering results are generated *offline* (see Section 2.6). To this extent, and following the taxonomy of (Han, Kamber, and Pei 2006), there are proposals that involve popular *partitioning*, *hierarchical*, *grid-based*, *density-based* and *model-based* methods. However, the SOM, which falls into the model-based category of algorithms, has not been properly explored in this context, establishing a gap in the current panorama of proposed methods (see Section 2.7). As shall be demonstrated, e.g., in Section 2.4.7, the SOM is very attractive for purposes of visual and exploratory knowledge discovery — one of the main reasons for its popularity. Favoring visual techniques, exploring and analyzing the vast volumes of data is becoming increasingly difficult. Information visualization and visual data mining can help to deal with the flood of information. The advantage of visual data exploration is that the user is directly involved in the data mining process (Keim 2002). In fact, a recent report (White 2013) found that “at organizations that use visual discovery tools, 48 percent of BI users are able to find the information they need without the help of IT staff”. Without visual discovery, the rate drops to a mere 23 percent. Also, managers using visual data discovery were 28 percent more likely than peers without visualized data to find timely information, according

to the study.

Finally, ubiquitous environments predict large amounts of streaming data being generated across spatially scattered devices. In such environments it is also not viable nor scalable to centralize the data streams and apply traditional clustering algorithms. Consequently, current approaches focus on the trade-off between local and global models, i.e., each node maintains its local model from data it senses or produces; then, global models are obtained by centralizing the local models and producing global clustering results, achieving a more scalable *distributed* mining environment (Rodrigues and Gama 2014). Also, applications scenarios related to, e.g., participatory sensing (Dutta et al. 2009), push the need to “share” models between nodes, through opportunistic networks, establishing *collaboration*. Hence, ubiquitous environments imply switching from one-shot learning tasks to a lifelong and spatially pervasive perspective (Gama 2010). These scenarios motivate the relevance of introducing new methodologies that allow the use of the SOM in ubiquitous environments, so as to benefit from the same type of exploratory cluster analysis performed at local nodes.

### 1.3 Contributions

The contributions presented in this thesis address first, and foremost, the underlying problem of exploratory cluster analysis (observations and features) by Self-Organizing Maps over individual multi-dimensional data streams (at local nodes) — *Problem 1* (Section 1.1). Then, after providing evidence that local nodes can produce SOM models from their own acquired data (through experimental evaluation), methodologies for collaborative and distributed learning strategies exploring the ubiquitous aspect of data streams are addressed, towards *Problem 2*.

Requirements for SOM variants to produce clustering results from data streams, and their intrinsic problems, are put forward by investigating the general limitations of the standard SOM algorithm and specific limitations of other SOM variants (that are able to learn incrementally). Based on this and on a literature review of current state-of-the-art methods, the research gap is identified and research paths established.

Concerning methods, inspired by the popular two-phase approach to data streams, a purely ANN-based methodology is presented based on a novel constrained variant of an *Adaptive Resonance Theory* (ART) algorithm, called *StreamART2A*, that is responsible for maintaining an *online* abstraction of the data stream through *micro-categories*; from these micro-categories *offline* SOM models are then obtained on-demand for exploratory cluster analysis. This approach is referred to as the *StreamART2A/SOM* methodology and includes also the proposal and empirical validation of a new error assessment metric. This metric regards quantifying the fit of a codebook to the evolving data stream and can be later used to develop change detection mechanisms. More importantly, it served as

the foundation for the proposal of a new SOM variant for data streams.

The StreamART2A/SOM methodology is found to work well across performed evaluations. However, it still induces a delay in obtaining SOM models for cluster analysis (obtained *offline*). Hence, a novel SOM variant tailored for non-stationary data streams is presented, i.e., the *Ubiquitous Self-Organizing Map* (UbiSOM). It relies on global assessment metrics of fit, i.e., between the codebook and the underlying distribution, to guide the evolution of the learning parameters towards stable representations during stationary phases and increasing the plasticity of the network during non-stationary phases of the data stream. This contrasts with the StreamART2A/SOM methodology in the sense that a SOM model is always available throughout the data stream, establishing this proposal as interesting for, e.g., real-time monitoring applications.

Both previous methods can be used for clustering of observations and clustering of features (intimately related to time series clustering), further establishing these proposals as versatile regarding existing methods. In respect to *feature clustering*, an automatic clustering method is presented to relief the user from (or aid in) visual inspection.

Leveraging the capabilities of the UbiSOM, collaborative and distributed learning strategies methodologies are put forward to tackle ubiquitous data streams. These address the current trade-off between local and global models, by maintaining individual local UbiSOM models that can be shared with other nodes (which additionally contain a separate global UbiSOM model) and/or centralized to produce a global SOM model.

Finally, a set of real-world problems are presented which exemplify application scenarios to which the proposed methods are applicable. A body of contributed software allows similar applications to be readily deployed into the real world.

In summary, the main contributions made in this thesis are the following:

- i. Requirements SOM variants should target when addressing data streams;
- ii. The *StreamART2A/SOM* methodology — a two-phase approach for clustering data streams, purely based on ANN methods, that target the SOM as the offline cluster algorithm; the online abstraction is performed by a variant of an ART algorithm;
- iii. The *Ubiquitous Self-Organizing Map* (UbiSOM) algorithm — a novel SOM variant tailored for non-stationary data streams. This algorithm continuously adapts to the underlying distribution of the data stream and allows real-time exploratory cluster analysis;
- iv. Collaborative and distributed learning methodologies that leverage the UbiSOM in ubiquitous environments;
- v. A set of real-world applications for contributions in *ii.*, *iii.*) and *iv.*);
- vi. Software framework and general purpose library for the UbiSOM algorithm and related methodologies.

These main contributions are supported by other minor contributions presented throughout this manuscript.

### 1.3.1 Publications

Most of the above contributions were initially presented or proposed in several publications, namely:

- Silva, B. and Marques, N. C. (2010). *“Ubiquitous data-mining with self-organizing maps”*. In: Workshop on Ubiquitous Data Mining in conjunction with the 19th European Conference on Artificial Intelligence (ECAI 2010), pp. 45–50.
- Silva, B. and Marques, N. C. (2010). *“Feature Clustering with Self-organizing Maps and an Application to Financial Time-series for Portfolio Selection”*. In: Proceedings of International Conference of Neural Computation, pp. 301–309.
- Silva, B. and Marques, N. C. (2012). *“Neural Network-based Framework for Data Stream Mining”*. In: Proceedings of the Sixth Starting AI Researchers’ Symposium (ECAI 2012). Vol. 241. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 294–305.
- Silva, B., Marques, N. C., and Panosso, G. (2012). *“Applying neural networks for concept drift detection in financial markets”*. In: Workshop on Ubiquitous Data Mining in conjunction with the 20th European Conference on Artificial Intelligence (ECAI 2012), pp. 43–47.
- Silva, B. and Marques, N. C. (2013). *“Ubiquitous Self-Organizing Maps”*. In: Proceedings of the 3rd Workshop on Ubiquitous Data Mining co-located with the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), pp. 54–55.
- Silva, B. and Marques, N. C. (2015). *“Ubiquitous Self-Organizing Map: Learning Concept-Drifting Data Streams”*. In: New Contributions in Information Systems and Technologies: Volume 1. Ed. by A. Rocha *et al.* Springer International Publishing, pp. 713–722.
- Silva, B. and Marques, N. C. (2015). *“The ubiquitous self-organizing map for non-stationary data streams”*. Journal of Big Data, 2(1), pp. 1–22.
- Marques, N. C. and Silva, B. and Santos, H. (2016). *“An Interactive Interface for Multi-dimensional Data Stream Analysis”*. In: Proceedings of the 20th International Conference Information Visualisation (IV), pp. 223–229.

## 1.4 Personal Note

This thesis is the culminate of approximately 14 years working with *Self-Organizing Maps*. After learning the basics and implementing the classical *Online SOM* algorithm in college,

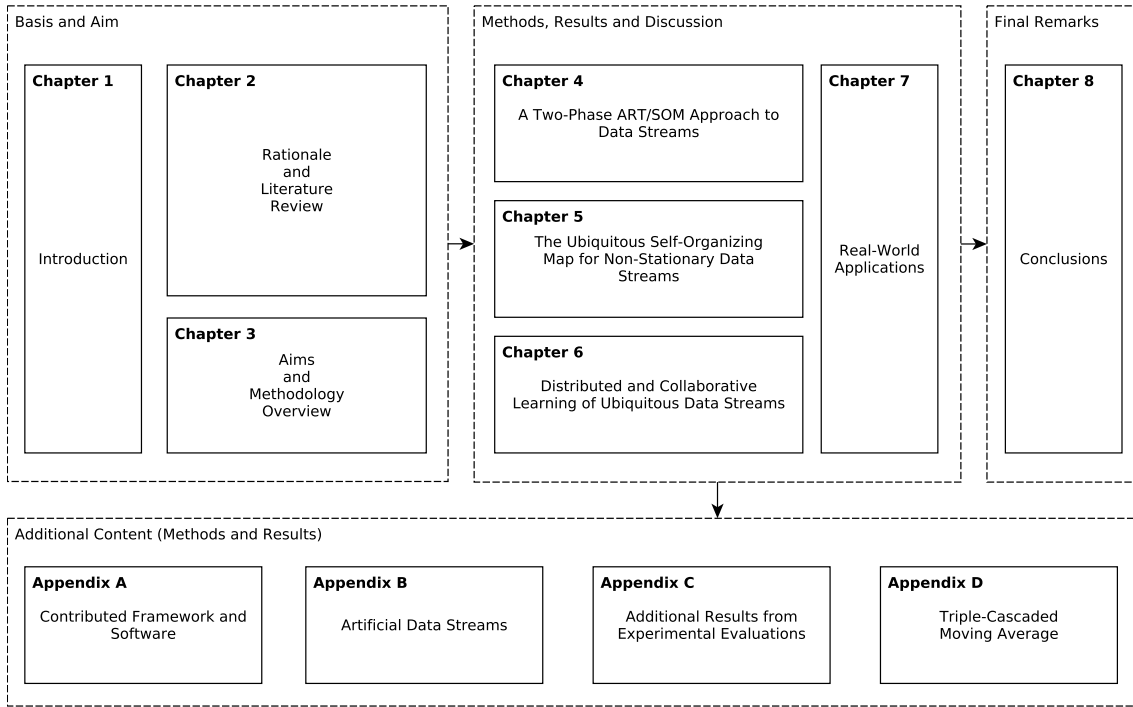


Figure 1.2: Thesis structure.

research involving the SOM initiated during my master's program in which I introduced a new parallelization method for the *Batch* SOM algorithm (Silva and Marques 2007), specifically tailored for large maps and *Big Data*. During these years, a deep understanding of the original algorithms, their strengths and weaknesses was obtained.

With the advent of the *Internet of Things* and current paradigm shift to ubiquitous computing, exploring the use of *Self-Organizing Maps* in streaming and ubiquitous environments seemed relevant and challenging. The intersection between *artificial neural networks*, in general, and *ubiquitous data mining* was practically unexplored. On the positive side, there was a lot of room for exploration and to make contributions; on the other hand, I became a bit overloaded between all the different contributions, evaluations, application scenarios and software development, in parallel with teaching. Consequently, this research study took longer than anticipated. Nonetheless, as in any doctoral program, this experience was extremely enriching both in academic terms as well as in my personal development.

## 1.5 Thesis Structure

The thesis is divided into eight chapters and four appendices, as schematized in Figure 1.2.

### 1.5.1 Basis and Aim (Chapters 1, 2 and 3)

The present chapter introduced the problem, providing some insight on its motivation, and summarized the contributions of this thesis. Chapter 2 presents the rationale for this thesis, namely regarding the SOM, challenges and motivating aspects of ubiquitous data streams, together with literature reviews: limitations of the standard SOM algorithms for data streams are highlighted and general requirements for data stream oriented variants are put forward; also, a survey of available variants regarding these requirements is presented and discussed; additionally, current state-of-the-art methods proposed for clustering data streams are reviewed, as well as proposals to tackle their ubiquitous aspect; finally, the previous exposure is discussed and research paths established, together with additional research questions. These later research questions are the main focus of Chapter 3, additionally stating their aims, respective objectives and a basic summary of the methodology proposed to solve them, enabling a focused reference for the topics addressed in the following chapters. Furthermore, some considerations and assumptions on the normalization of data streams are discussed. Finally, a brief summary of artificial data streams, used in the experimental evaluation of the forthcoming proposals, is provided. These three chapters form the basis for the development of the thesis.

### 1.5.2 Methods, Results and Discussion (Chapters 4, 5, 6 and 7)

These can be regarded the main chapters of this thesis, in the sense that they expose the main contributions. They concentrate the author's answers to the research questions enunciated in the previous chapter. Chapter 4 presents and evaluates the *StreamART2A/SOM* methodology. Chapter 5 introduces the *UbiSOM* algorithm and the automatic feature clustering method. Chapter 6 then addresses the ubiquitous aspect of data streams with the *UbiSOM*, presenting distributed and collaborative learning methodologies. Chapter 7 exemplifies real-world applications of the algorithms and methodologies presented in the previous three chapters.

### 1.5.3 Final Remarks (Chapter 8)

This research is not without limitations. In Chapter 8 main findings are discussed, by revisiting the initial research questions and summarizing the contributions. Strengths and limitations of the contributions are discussed, together with recommendations for future work.

### 1.5.4 Appendices

This thesis has four additional appendices. Appendix A presents contributed software that can be readily used to replicate the real-world application scenarios of Chapter 7 in practice. Appendix B describes in more detail the artificial data streams used in the experimental evaluations of Chapters 4, 5 and 6. Appendix C contains detailed and/or

additional experimental results regarding Chapters 4 and 5. Finally, Appendix D provides additional information of the *triple-cascaded moving average* that approximates the behavior of a *Gaussian* filter, which is used in the UbiSOM algorithm.





# Rationale and Literature Review

I have no special talents. I am just passionately curious.

---

ALBERT EINSTEIN, GERMAN PHYSICIST (1879-1955)

A deeper understanding of the problematic addressed in this thesis is necessary as well as establishing the relevancy of the undergone research comparatively to existing approaches.

## 2.1 Chapter Overview

This chapter covers the necessary aspects needed to comprehend the established problem, the research context, its motivating aspects regarding current approaches and, consequently, how the solutions can improve the state-of-the-art.

The chapter is organized as follows. Motivation and aims are established in Section 2.2. Section 2.3 addresses the challenges, relevance and motivating scenarios regarding cluster analysis over ubiquitous data streams. Section 2.4 provides a deep understanding of the SOM algorithm and its limitations regarding data streams. The exploratory cluster analysis abilities of the SOM are also compared and contrasted with other popular clustering methods. In Section 2.5 requirements SOM variants should address in dealing with data streams are put forward, together with a survey and evaluation of existing variants against these requirements. Current methods for cluster analysis on data streams are surveyed in Section 2.6. Section 2.6.3 reviews some current strategies in dealing with ubiquitous data streams. Finally, Section 2.7 provides a critical analysis of the current landscape to further motivate the use of the SOM, summarizing the answers to the initial research questions and providing research directions.

## 2.2 Motivation and Aim

Despite being a three decade old algorithm, Kohonen's *Self-Organizing Map* algorithm (Kohonen 1982, 2001) has proven throughout literature to be a powerful tool for unsupervised knowledge discovery, namely exploratory cluster analysis, across all scientific domains (Pöllä, Honkela, and Kohonen 2009). However, no works are found regarding its clear applicability to the intrinsic problems of data streams and their distributed nature.

The aim of this chapter is to provide answers to the initial research questions put forward in Section 1.1, so as to clearly motivate and contextualize the undergone research. Also, from this exposition, to lay out research paths that raise additional research questions.

## 2.3 Unsupervised Knowledge Discovery from Data Streams

In the past 50 or so years, information technology, together with the fast development of powerful data collection and storage tools, led to an explosion of data gathered from business, society, science and engineering, medicine and almost every other aspect of everyday life. The human inability to process and make sense of these vast amounts of data quickly pushed the need to develop tools to automatically uncover valuable information from such data into organized knowledge. Consequently, the interdisciplinary field of *Knowledge Discovery and Data Mining* (KDD)<sup>1</sup> surged, has evolved, and continues to evolve, from the intersection of research fields such as machine learning, pattern recognition, databases, statistics, artificial intelligence, knowledge acquisition for expert systems, data visualization, and high-performance computing. The unifying goal is to extract high-level knowledge from low-level data in the context of large volumes of data. Many people use the terms *knowledge discovery* (from data) and *data mining* as synonyms, while others view data mining as merely an essential step in the process of knowledge discovery, i.e., the application of specific algorithms for extracting patterns from data (Han, Kamber, and Pei 2006).

From its inception KDD focused its attention in batch learning, extracting knowledge from *datasets*. By nature, a dataset consists of a fixed amount of data that can be revisited several times to extract relevant information. Nowadays, data streams (Aggarwal 2007; Gama 2010) are generated naturally within several applications, as opposed to simple datasets. Such applications include network monitoring, web mining, sensor networks, telecommunications, and financial applications, for example. All have vast amounts of data arriving continuously and in these applications it may not be feasible to store all the arriving data into a traditional database and/or file format.

---

<sup>1</sup>Originally the acronym stood for Knowledge Discovery from Databases. It was then changed to define knowledge discovery from a wider range of data sources.

More formally, a data stream  $\mathcal{S}$  is a massive sequence of observations  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ , i.e.,  $\mathcal{S} = \{\mathbf{x}_i\}_{i=1}^N$ , which is potentially unbounded ( $N \rightarrow \infty$ ). Each observation is described by an  $d$ -dimensional feature vector, i.e.,  $\mathbf{x}_i = [x_i^j]_{j=1}^d$ , belonging to a feature space  $\Omega$  that can be continuous, categorical or mixed. Each feature describes an individual measurable property of the phenomenon being observed. In this work only continuous spaces are considered.

When no other information is available, knowledge discovery methods are usually based on unsupervised learning, e.g., *cluster analysis* or simply *clustering*. Clustering can be defined as a data mining task which aims at finding meaningful subsets of observations (clusters) from a larger set of observations (Han, Kamber, and Pei 2006). The members of a cluster are more like each other than they are like members of other clusters, based on some similarity measure. From a higher-level standpoint, clustering is a data segmentation task and is useful in understanding the underlying structure of the data. Another type of clustering, comparatively to this *clustering of observations*, is the *clustering of features*. Feature clustering consists in arranging features into homogeneous clusters, i.e., groups of features which are strongly related to each other and thus bring the same information. Besides providing additional understanding of the data, such approaches can then be useful for dimensionality reduction and feature selection (Dash and Liu 2000). In the context of data streams, clustering of features is closely related to time-series clustering (Liao 2005) — see Section 2.6.2. Therefore, the problem of cluster analysis (regardless of its type) over data streams concerns maintaining a clustering result that can be presented to the user at any time.

### 2.3.1 The Data Stream Model

In the context of static data a plethora of algorithms have been proposed to address the clustering problem (Berkhin 2006). However, static data can be revisited several times to build clustering models, processing time is not a huge issue and it assumes the underlying distribution is stationary, i.e., the underlying structure is immutable. On the other hand, data streams exhibit some characteristics that limit the applicability of traditional clustering algorithms. In (Babcock et al. 2002) the *data stream model* is presented, highlighting the difficulties algorithms are presented with, comparatively to static data:

- Observations in the data stream arrive online and algorithms have no control over the order they are processed;
- Data streams are potentially unbounded in size;
- Once an observation has been processed it is discarded or archived — it cannot be retrieved easily unless it is explicitly stored in memory, which is typically small

relative to the size of the data streams. In the case of *transient* data streams, past observations cannot be recovered.

Consequently, considering their nature, cluster analysis over data streams should address the following challenges (Barbará 2002):

- **Limited time:** data streams arrive continuously, which requires fast and real-time response. Therefore, the clustering algorithm should be able to handle the speed of data streams in limited (constant) time. Ideally, data stream clustering algorithms should only require one-pass over the data;
- **Limited memory:** data streams are potentially unbounded, which would require unlimited memory. However, the clustering algorithm must operate within memory constraints, providing a *compact model* of the clustering;
- **Evolving data:** the algorithm must consider that data streams evolve considerably over time, e.g., new clusters might appear, others disappear, reflecting the dynamics of the stream. This must also be addressed in the light of limited memory;
- **Noisy data:** any clustering algorithm must be able to deal with random noises present in the data, since outliers have great influence on the formation of clusters;
- **High-dimensional data:** some data streams are high dimensional in their nature. Consequently, the clustering algorithm has to overcome this challenge.

Although these requirements are only partially fulfilled in practice, it is instructive to keep them in mind when designing algorithms for clustering data streams. One thing that is agreed is that such algorithms can only provide approximate results, since data cannot be revisited (Guha et al. 2003).

Evolving (non-stationary) data is a big challenge to algorithms. This means the underlying distribution, from where clustering is being performed, may change over time, after some minimum permanence. In some literature this is defined as *concept drift* (Gama 2010). A way to formalize this is to consider a data stream  $\mathcal{S}$  as a sequence of stationary distributions  $\mathcal{D}_i$ , i.e.,  $\mathcal{S} = \{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_\infty\}$ , however without any knowledge of the size of each  $\mathcal{D}_i$ . Nonetheless, the evidence of change is reflected in some way in the observations, e.g., distributions  $\mathcal{D}_i$  and  $\mathcal{D}_{i-1}$  have different mean, variance or correlations between features. Also, old observations become irrelevant and clustering models must discard that information. However, a difficult problem in handling change is distinguishing between true drift and noise. The difference between noise and observations of another distribution is *persistence*: there should be a consistent set of observations of the new distribution. Some algorithms may overreact to noise, misinterpreting it as concept drift, while others can be too robust to noise, adjusting to the real changes too slowly. An ideal learner should combine robustness to noise and sensitivity to concept drift (Gama

2010). Embedding change detection in the learning process is one of the most challenging problems when learning from data streams (Kifer, Ben-David, and Gehrke 2004).

There has been a lot of research regarding data stream clustering algorithms, mostly targeting clustering of observations. Section 2.6.1 provides an overview of such methods, which involve either adapting some traditional algorithms for single-pass strategies or, more usually, to combine an *online* summarization stage where the data stream is continuously “condensed” into summary structures; traditional clustering algorithms are then applied *offline* to the summary data. The distinction between clustering observations and clustering features is not an issue with static data, since observations and features can be easily transposed. However, in the context of data streams, the standard matrix transposition is not applicable since it can not be applied to continuously arriving data. Clustering features in data streams requires different algorithms, but not much work has been developed towards this — see Section 2.6.2.

Regarding the ubiquitous aspect of data streams, some of the following motivating scenarios shed light on the relevance of cluster analysis over ubiquitous data streams. Nonetheless, this specific problematic is addressed later in Section 2.6.3.

### 2.3.2 Motivating Scenarios and Applications

Examples motivating the clustering of data streams can be found in many application domains, including finance (Kargupta et al. 2002; Kontaki, Papadopoulos, and Manolopoulos 2008), web applications (Antonellis, Makris, and Tsirakis 2009), networking (Zhang and Wang 2010), and sensor monitoring (Rodrigues, Gama, and Lopes 2008; Vatsavai et al. 2010). Also, the current computing paradigm shift that we are witnessing nowadays towards ubiquitous computing, e.g., Internet of Things (IoT), is intrinsically related to data streams. Ubiquitous computing encompasses devices such as intelligent sensors, vehicle control systems, household appliances, computer peripherals and other embedded computer systems, all which can produce massive amounts of data in the form of ubiquitous data streams. Relating to knowledge discovery, this has led to the emergent field of *Ubiquitous Data Mining* (UDM). UDM is devoted to knowledge discovery from data on mobile, embedded and ubiquitous devices and the processing networks they form (Gaber et al. 2014). In ubiquitous environments, devices often have the ability to move and they never see the global picture — they know only their local spatial-temporal environment. However, the device may be able to exchange information with other devices, thus forming a truly distributed environment.

Some examples of the above applications are:

- Clustering data streams collected by individual sensors or within sensor networks (Gama, Rodrigues, and Lopes 2011; Rodrigues, Gama, and Lopes 2008; Vatsavai et al. 2010) is a typical application. Sensor networks may be responsible, e.g., for

measuring ambiance and pollutant concentrations in a given city. In this context, data is collected from a set of sensors distributed all around the city. Clustering this kind of information can help to understand patterns of pollution over different periods of the day across a particular place, area or in the entire city. A similar application can be performed over water distribution networks, e.g., in (Li et al. 2011);

- Nowadays, financial applications are demanding increasingly timely knowledge, as trading orders are put forward every second. The continuous variations of the prices of a set of financial assets can be regarded as a multivariate data stream, composed by individual time series. Clustering of the time series is one of the learning tasks required in this scenario, considering that it allows the identification of similar and dissimilar financial assets along trading hours, which can be useful for portfolio management or placing trading orders;
- Some proposed systems also illustrate additional applications:
  - *MobiMine* (Kargupta et al. 2002) is credited to be the first data stream mining system. It is a client/server PDA-based distributed data mining application for financial data streams. The server collects data from different financial websites, selects active stocks and applies online data mining algorithms to the stock data. On the client side the user can perform his portfolio management. Computations regarding the impact of global active stocks in the local portfolios are performed in the server and results visualized in the PDAs; the system minimizes transferred data applying Fourier transformations. It should be noted that the server performs all data mining tasks.
  - Another system is the *Vehicle Data Stream Mining System* for vehicle monitoring (Kargupta et al. 2004). It is a ubiquitous data stream mining system that allows continuous monitoring and pattern extraction from data streams generated on-board a moving vehicle. The mining component is located on a PDA. The system may also interact with the control station to alert the network or improve its model.
  - In (Stahl et al. 2010) the term *Pocket Data Mining* is proposed to describe collaborative mining of streaming data in mobile and distributed computing environments, where authors proposed a general framework based on software agents.

Being able to produce clustering models in real-time assumes great importance within all these applications.

## 2.4 The Self-Organizing Map

The Self-Organizing Map (SOM) (Kohonen 1982, 2001) is an *artificial neural network* (ANN) clustering model that excels in data visualization. ANN-based clustering is performed through *competitive learning* schemes, which are traced back to early works of Rosenblatt (Rosenblatt 1961), von der Malsburg (Malsburg 1973) and Grossberg (Grossberg 1976). Some preliminary considerations on competitive learning are given in the next section before delving into the SOM itself. This should allow a deeper understanding of the problems data streams impose on competitive learning schemes.

### 2.4.1 Competitive Learning Fundamentals

Competitive learning is a form of unsupervised learning in artificial neural networks, in which neurons compete among themselves to represent a subset of the input data. In ANN terminology, observations are called *input patterns*. With static data, the input patterns  $\{x_1, x_2, \dots, x_N\}$  form the *training dataset*  $\mathcal{D}$ , where  $N$  is the total number of observations. When describing learning procedures, the concept of (discrete) *time* is recurrent. The current time is referred to as  $(t)$  and the next time step as  $(t + 1)$ . Usually, at each time step an input pattern  $x(t)$  will be presented to the network. If presenting the training dataset several times, each presentation is called an *epoch*.

According to (Rumelhart and Zipser 1985), a competitive learning scheme consists of the following three basic components:

- i. “Start with a set of units that are all the same except for some randomly distributed parameters which makes each of them respond slightly differently to a set of input patterns;
- ii. Limit the strength of each unit;
- iii. Allow the units to compete in some way for the right to respond to a given subset of inputs.”

Competitive learning can be implemented using a completely linked two-layer feed-forward neural network as shown in Figure 2.1. The nodes (neurons) in the *input layer* admit input patterns that are fully connected to the output nodes in the *competitive layer*. Each output node is associated with a prototype vector  $w_k$ , with  $k = 1, \dots, K$ , where  $K$  is the number of prototypes, stored in terms of synaptic weights  $w_{ki}$ , with  $i = 1, \dots, d$ , representing the connection between input node  $i$  and output node  $k$ , i.e., the prototypes have the same dimensionality as the input patterns. Some competitive learning networks can impose lateral connections between units in the competitive layer to promote cooperation or inhibition, e.g., the SOM, as detailed in Section 2.4.2.

In simple competitive learning, given an input pattern  $x(t)$ , randomly initialized neurons in the competitive layer compete with each other to represent the input, and only the winner neuron  $c$  becomes activated or “fired”. The neuron who wins the competition



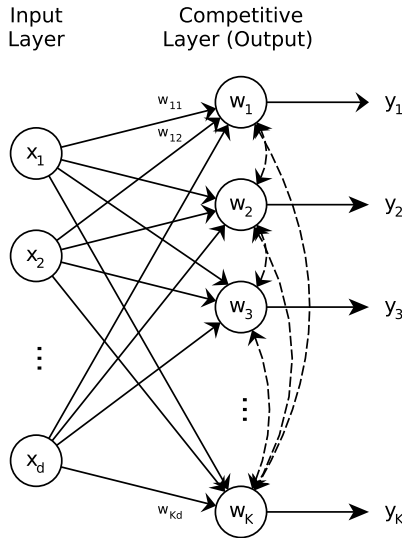


Figure 2.1: Architecture of the competitive learning network. The output selects one of the prototype  $w_c$  by setting  $y_c = 1$  and all  $y_j = 0$ ,  $j \neq c$ .

is the one which is found to be more similar to the input pattern. If using a distance function, e.g., the Euclidean distance ( $\| \cdot \|$ ), to measure dissimilarity, then the winning neuron  $c$  is determined by Eq. (2.1), i.e., the one which has the “closest” prototype to the input pattern.

$$c = \min_k \| \mathbf{x} - \mathbf{w}_k \| \quad (2.1)$$

While performing clustering, competitive learning aims at minimizing the *mean squared error* (MSE) cost function:

$$MSE = \frac{1}{N} \sum_{t=1}^N \| \mathbf{x}(t) - \mathbf{w}_c(t) \|^2. \quad (2.2)$$

Targeting this criteria, competitive learning imposes a gradient-descent prototype vector update towards the input pattern following the competitive learning update rule in Eq. (2.3):

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \eta(t)[\mathbf{x}(t) - \mathbf{w}_c(t)] \quad (2.3)$$

where  $0 < \eta(t) \leq 1$  is the *learning rate*. Generally, the learning rate decreases monotonically over time, i.e., an *annealing* scheme. For example, one can select  $\eta(t) = \eta_i(1 - \frac{t}{N})$ , where  $\eta_i$  is the initial learning rate. The learning rate determines the adaptation of the prototype towards the input pattern and is directly related to the *convergence*. If  $\eta$  equals zero, there is no learning. If it is set to one, it will result in fast learning and the prototype vector is directly pointed to the input pattern. For other choices of  $\eta$ , the new position of the prototype vector will be on the same “line” between the old prototype and the



input pattern. The rationale regarding annealing schemes for the learning rate is to allow prototypes to slowly converge to their optimal positions, minimizing the MSE cost function.

This competitive learning paradigm only allows learning for a particular winning neuron and is called *winner-take-all* (WTA) or *hard competitive learning* (Fritzke 1997). Note that, depending on the initialization of the prototypes, some neurons may never win the competition — called “dead-units”. On the other hand, learning can also occur in a cooperative way, meaning that not only the winning neuron adjusts its prototype, but all other prototypes may be adjusted depending on how approximate they are to the input pattern. This is called *winner-take-most* (WTM) or *soft competitive learning* (Fritzke 1997), from which the SOM is a prominent example. These algorithms are less likely to be trapped at local minima and to generate dead units than hard competitive alternatives (Baraldi and Blonda 1999).

Competitive learning in the presented form, despite being a computationally efficient procedure — time and space complexity of  $O(NK)$  and  $O(K)$ , respectively, is not particularly suited for data streams. Data streams, by assuming  $N \rightarrow \infty$ , limit the applicability of annealing schemes. Constant values for the learning rate are used only in specific models (see ART networks in Section 2.4.1.2) because they also have problems. If we keep  $\eta$  fixed, a learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large, although allowing to rapidly adjust to non-stationary data, can hinder convergence and cause prototypes to fluctuate around or to even diverge.

#### 2.4.1.1 Relation to k-means

There is an intimate relationship between the WTA competitive learning scheme and the incremental *k-means* (MacQueen 1967) algorithm. The *k-means* algorithm is also based on a fixed number of data prototypes (centroids) and both suffer from poor prototype (centroid) initialization, if randomly chosen. Let  $\mathbf{z}_k \in R^d$  be a *k-means* centroid. The incremental *k-means* algorithm works by adjusting the closest centroid  $\mathbf{z}_c$  to an observation  $\mathbf{x}(t)$  by  $\mathbf{z}_c(t+1) = \frac{n_c \mathbf{z}_c(t) + \mathbf{x}(t)}{n_c + 1}$ , where  $n_c$  is the number of observations associated with that centroid. In a WTA scheme this can be approximated by allowing each neuron to have its own learning rate, e.g., an annealing scheme such as  $\eta_c = \frac{1}{n_c}$  (Yair, Zeger, and Gersho 1992).

Again, if we consider non-stationary data, as time passes each centroid/prototype loses the ability to perform large updates and to properly converge to a new distribution, because the learning rate is continuously getting smaller. We can say that while the procedure achieves *stability* over a stationary distribution, it loses *plasticity* over time.

### 2.4.1.2 ART networks

*Adaptive Resonance Theory* (ART) (Grossberg 1976), contrary to popular believe, is not an ANN model, but a whole theory on aspects of how the brain processes information. The theory led to an evolving series of real-time unsupervised network models for clustering, pattern recognition, and associative memory (Carpenter and Grossberg 1988; Carpenter, Grossberg, and Reynolds 1991; Carpenter, Grossberg, and Rosen 1991b), some more complex than others, which we call ART networks.

ART networks have the ability to adapt, yet not forget past information, and this is what Grossberg refers to as the *stability–plasticity dilemma* (Grossberg 1976), somewhat tackling non-stationary data. What differentiates the competitive learning scheme of ART networks is that neurons can be added to the network dynamically. ART2-A (Carpenter, Grossberg, and Rosen 1991a) networks are extensions of the original ART2 network (Carpenter and Grossberg 1988) to handle continuous real-valued features. During learning, the stored prototype of a *category*<sup>2</sup> is adapted only when an input pattern is *sufficiently similar* to it; otherwise, a new category is created with the input pattern as its initialization. The meaning of being sufficiently similar is dependent on a *vigilance* parameter  $\rho \in [0, 1]$ . If  $\rho$  is large, the similarity condition becomes stringent and many finely divided categories are formed. In contrast, a smaller  $\rho$  gives a coarser categorization, resulting in fewer categories.

In the ART2-A algorithm, starting with an empty set of prototypes, each input pattern  $x(t)$  is compared to the  $k$  stored prototypes in a WTA fashion. If the degree of similarity between the current input pattern and the most similar category  $w_c$  is at least as high as the vigilance parameter  $\rho$ , this category is chosen to represent the input and its prototype is adapted by the previous competitive update rule in Eq. (2.3). Otherwise, a new category is created, where the example is used as the prototype initialization. Due to its dynamic nature, ART2-A uses a constant learning rate  $\eta \ll 1$ , chosen to prevent prototype  $w_c$  from moving too fast and therefore destabilizing the learning process.

The vigilance parameter  $\rho$  imposes a *perceptive field* around the categories; if using the Euclidean distance to measure dissimilarity, this can be seen as an hyper-sphere around a prototype, acting as a distance threshold to decide if the closest category should be updated or a new dynamically created.

This competitive learning scheme may seem more adequate to data streams, since the ART2-A network maintains an indefinite *plasticity* over time. However, estimating  $\rho$  is not easy. A high vigilance parameter can lead to the creation of numerous categories, which violates the model compactness requirement (recall Section 2.3.1). On the other hand, a low vigilance parameter can, in the worst-case scenario, result in only one category. Even with a properly estimated  $\rho$  value, the non-stationary characteristic of data streams could lead to the unbounded growth of the number of categories. Also, and

<sup>2</sup>In ART literature a cluster centroid is frequently referred to as a *category*.

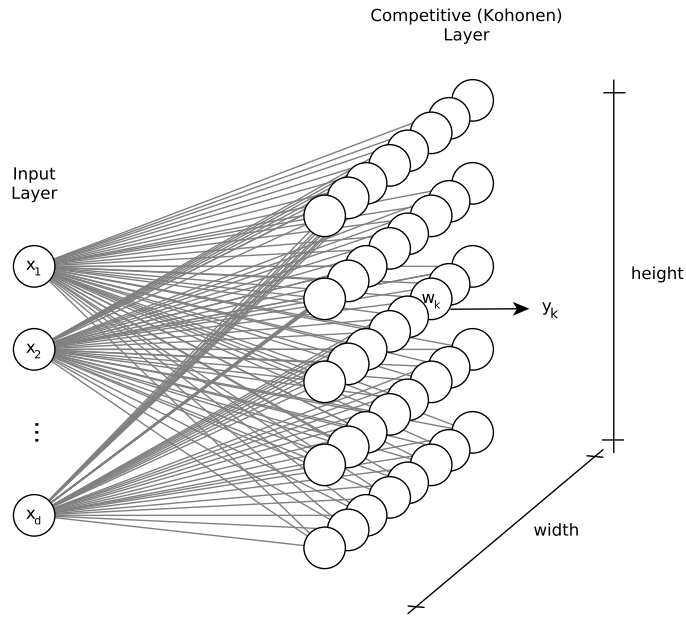


Figure 2.2: Structure of a  $8 \times 5$  SOM network. It follows the same structure as the competitive learning network, but with a two-dimensional output layer where lateral connections impose cooperation among neurons.

more importantly, the *stability* aspect means that old information is not discarded in the presence of change.

### 2.4.2 SOM Model and Algorithm

The SOM is a *topology-preserving* competitive learning model, based on a WTM strategy — soft competitive learning. It is also biologically inspired by the way the brain organizes itself, in the sense that different parts of the brain specialize in different sensory inputs. The SOM network has the same structure as the competitive learning network and lateral connections impose *cooperation* among neurons. The SOM is a one-, two-, or higher-dimensional lattice of neurons, albeit the two-dimensional case is the most common, due to the data visualization aspect, introduced later in Section 2.4.5. Figure 2.2 depicts a two-dimensional SOM network, where each unit in the *Kohonen* (competitive) layer contains a prototype vector  $w_k$ , as previously defined. Neurons are organized into a rectangular *lattice* of size  $K = width \times height$ . To enable data visualization, the SOM must use a higher number  $K$  of prototypes than a predefined number of desired clusters, as in k-means (see Section 2.4.3).

The classical SOM algorithm, referred hereafter as the *Online* SOM algorithm, proceeds iteratively over time. For each observation  $x(t)$ , the *best matching unit* (BMU), denoted by  $c$ , is determined likewise by Eq. (2.1). Next, the prototype vectors are updated in a WTM fashion: the BMU and its *topological neighbors* are moved towards the example in the input space by the *Kohonen* learning rule, which is a major development in

competitive learning:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta(t) h_{ck}(t) [\mathbf{x}(t) - \mathbf{w}_k(t)] \quad (2.4)$$

where:

$t$  iteration;

$\eta(t)$  learning rate;

$h_{ck}(t)$  neighborhood kernel centered on the BMU.

The winning neuron locates the center of a topological neighborhood<sup>3</sup> of cooperating neurons. Therefore, based on this observation, the topological neighborhood should decrease with lateral distance. The *Gaussian* function is often taken for the neighborhood function — Eq. (2.5), because it exhibits this behavior and is independent of the location of the winning neuron:

$$h_{ck}(t) = e^{-\frac{\|r_c - r_k\|^2}{\sigma(t)^2}} \quad (2.5)$$

where  $r_c$  and  $r_k$  are positions of units  $c$  and  $k$  on the SOM lattice. The parameter  $\sigma(t)$  establishes the width of the topological neighborhood and decreases monotonically over time, as well as  $\eta(t)$ . The monotonic decrease of these learning parameters is a critical condition to achieve convergence and any decreasing function, e.g., annealing scheme, can be used (considerations on parameterization are presented in Section 2.4.4).

This process produces a topological ordering of the map, in the sense that adjacent neurons in the lattice will have similar prototype vectors that, overall, approximate the *probability distribution function* (PDF)  $p(x)$  of the underlying distribution. This implies that the prototype vectors will order themselves with approximately equal distances between them if input vectors appear with even probability throughout a section of the input space. If input vectors occur with varying frequency throughout the input space, the map tends to allocate neurons to an area in proportion to the frequency of input vectors there.

It should be noted that with very small datasets, the algorithm must endure through a predefined number of epochs to allow an initial global ordering and posterior convergence of the prototypes (this is addressed in Section 2.4.4). With data streams this is not an issue, since a large volume of observations is expected. Also, in data stream settings, it is assumed that the underlying process generating the observations will be in a persistent state throughout some period of time (Gama 2010).

---

<sup>3</sup>Neurobiologically, a neuron that is firing tends to excite the neurons in its immediate neighborhood more than those far away.

### 2.4.2.1 The Batch Algorithm

There is a *Batch* version of the SOM algorithm, where prototype updates are deferred to the end of a learning epoch. The batch update rule is given by:

$$\mathbf{w}_k(t_f) = \frac{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')\mathbf{x}(t')}{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')} \quad (2.6)$$

where  $t_0$  and  $t_f$  stand for the beginning and end of the current epoch, respectively. Consequently,  $\eta(t)$  and  $\sigma(t)$  are decreased monotonically by epoch. The BMU is computed from  $\mathbf{w}(t_0)$  within each epoch. If the unit  $k$  is not affected by any of the examples during a learning epoch, i.e., the unit is not a BMU nor a neighbor of any BMU, then  $\mathbf{w}_k(t_f) = \mathbf{w}_k(t_0)$ . By comparing Equations (2.4) and (2.6) the learning rate is implicitly given by:

$$\eta(t_0) = \frac{1}{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')} \quad (2.7)$$

and does not need to be parameterized as in the Online algorithm, thus eliminating a potential source of poor convergence (if badly estimated). Although being computationally faster, some authors (Fort, Letremy, and Cottrel 2002) point out that the quality of the clustering produced by the Batch algorithm is somewhat inferior to the one of the Online SOM algorithm. However, for the purposes of exploratory cluster analysis, this research did not find any significant differences that would hinder the use of the Batch variant.

### 2.4.2.2 Limitations on Data Streams

By keeping the number of prototypes fixed, the SOM suffers from the same problems as simple competitive learning schemes when dealing with data streams. Annealing procedures for the learning parameters cannot be devised when  $N \rightarrow \infty$  and as time progresses the network loses the ability to “learn” new distributions arising from the data stream, i.e., loses *plasticity*. Clearly, the Batch algorithm is not suited for data streams, since it can only be applied when the whole data is present.

### 2.4.3 Related Algorithms

The SOM algorithm is related to some other algorithms and methods, as discussed in the following paragraphs:

**k-means.** As discussed in Section 2.4.1.1, the k-means clustering algorithm is closely related to competitive learning and is a special case of the SOM. By setting the width  $\sigma$  of the neighborhood kernel  $h_{ck}$  to zero, the SOM degenerates in a WTA scheme, similar to the incremental k-means (Kohonen 2001). However, in (Baao, Lobo, and Painho 2005a),

it is shown that a one-dimensional SOM can outperform k-means clustering, since the SOM is less sensitive to local *minima* (due to the WTM strategy).

Despite this close relation, the best way of using both algorithms in data mining is different. Whereas in the k-means clustering algorithm the number of  $K$  clusters should be chosen according to the number of clusters that are present in the data, in the SOM the number of prototype vectors should be chosen to be much larger, irrespective of the number of clusters (Ultsch 1995). Doing so, the cluster structures and descriptions can become visible by means of special visualization techniques that are presented in Section 2.4.5.

**Vector Quantization.** Vector quantization (VQ) is a classical method for approximating a PDF of the vector variable  $\mathbf{x} \in R^d$  by using a finite number of vector prototypes. The set of prototypes  $\{\mathbf{w}_1, \dots, \mathbf{w}_K\} \in R^d$  is referred to as the *codebook*. Approximation of  $\mathbf{x}$  is to find the prototype  $\mathbf{w}_c$  from the codebook that is closest to  $\mathbf{x}$ . This is the *nearest-neighbor* paradigm. The density matching property of VQ is powerful, especially for identifying the density of large and high-dimensioned data.

The SOM is a VQ method and the set of prototypes of the SOM is indeed frequently referred to as the *SOM codebook*. The SOM, however, is not an optimal VQ procedure due to the “tension” induced by neighboring relationships among prototypes.

One of the most well-known methods to perform VQ is the *LBG* algorithm (Linde, Buzo, and Gray 1980), which solves the prototype initialization issue by splitting an initial prototype centered in the input space until  $K$  prototypes are obtained. In (Yair, Zeger, and Gersho 1992) the SOM algorithm was adapted for optimal VQ by changing the learning scheme from WTM to WTA later in time. A similar approach is used in the *Neural Gas* (NG) algorithm (Martinetz, Berkovich, and Schulten 1993), however not imposing neighboring relations. In NG, the magnitude of the prototype updates is not based on a distance kernel  $h_{ck}$ , but ranked-based instead, sorting the prototypes based on their distance to the input pattern.

**Principal Curves.** Principal curves and surfaces (Hastie and Stuetzle 1989) are very similar to the SOM in their concept. The goal is to find the central curve (or surface) in the input space. Each point in the principal curve averages all the points that project to it. One way to create a principal curve is to apply a one-dimensional SOM to the multidimensional data (Wesolkowski 2002), since the SOM prototype vectors can be interpreted as conditional averages of the data.

**Vector Projection.** In vector projection (VP) algorithms the objective is to find low-dimensional coordinates for representing high-dimensional observations, while preserving the relationships among the input data. Thus, the goal is dimensionality reduction and visualization if the output space is two- or three-dimensional. Probably the most known vector projection algorithms are multi-dimensional scaling (MDS) (Kruskal 1964),



Sammon's mapping (Sammon 1969), Principal Component Analysis (PCA) and Curvilinear Component Analysis (CCA) (Demartines and Héroult 1997). By examining each error function used by these algorithms, they differ mainly in the importance given to local or global topology. While in MDS the global organization is emphasized, in Sammon's mapping and CCA more importance is given to local topology. The SOM is a vector projection algorithm because each pattern of the input space is projected into well defined position in a, e.g., two-dimensional map.

#### 2.4.4 Ordering, Convergence and Parameterization

The theoretical foundations of the SOM have been discussed thoroughly by (Cottrell, Fort, and Pagès 1998). Mathematical proof of ordering and convergence of the SOM algorithm has only been reported for the one-dimensional map (Erwin, Obermayer, and Schulten 1992) and in particular circumstances: only when the neighborhood function is *convex* (e.g., the Gaussian neighborhood function is). In higher dimensional maps no mathematical proof of ordering and convergence has been achieved, probably because the SOM does not minimize a known cost function (the neighborhood kernel makes it difficult to derive one), so estimation of its learning parameters have been based on empirical and numerous experimental results obtained over the years. Notwithstanding the absence of mathematical proof, the SOM algorithm has proved experimentally across literature that it is reliable if parameterized correctly (Pöllä, Honkela, and Kohonen 2009).

##### 2.4.4.1 Learning Phases

According to (Kohonen 2001), starting from a state of complete disorder, the SOM algorithm gradually achieves an organized representation of the input space, provided that the parameters of the algorithm are chosen properly. Also, the adaptation process of the algorithm should be decomposed into two phases: an *ordering* phase followed by a *convergence* phase:

**Ordering phase.** It is during this first phase of the adaptation process that the topological ordering of the prototype vectors takes place. This ordering phase is relatively short in comparison to the convergence phase. Large values for the neighborhood radius  $\sigma(t)$  and learning rate  $\eta(t)$  should be used, such that the prototype vectors initially take large steps all together toward the area of input space where input vectors are occurring — this can be thought as the “unfolding” of the map. These values then should decrease to their tuning values and encompass only the closest neighbors. During this phase the map tends to order itself topologically over the presented input vectors. Kohonen advises that the ordering phase should consist at least in the presentation of 1000 input patterns; at this point the network should be fairly well-ordered.

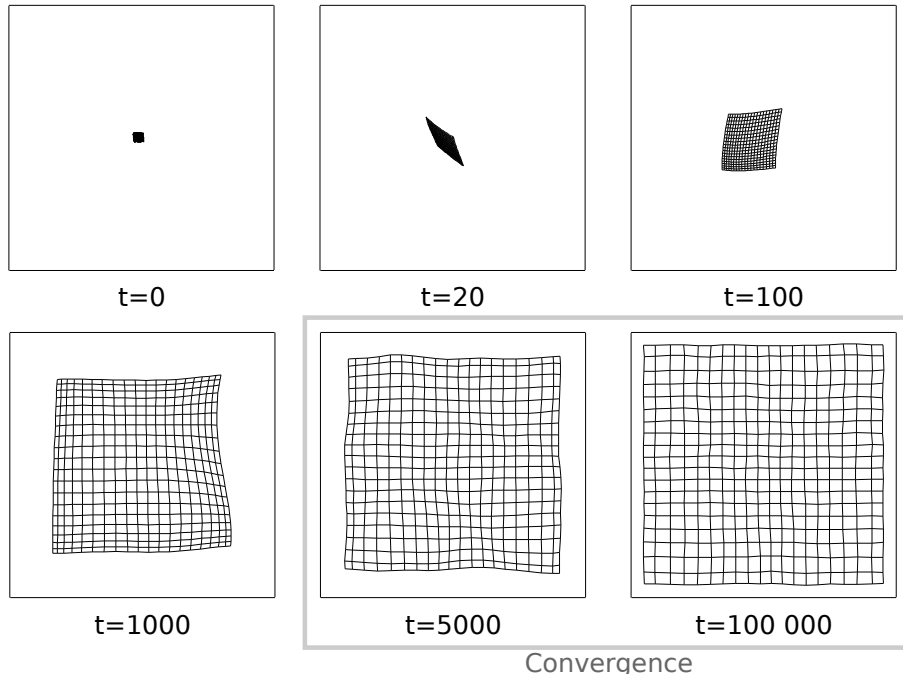


Figure 2.3: Ordering and convergence phases of the SOM over an uniform 2D distribution.

**Convergence phase.** This phase lasts for the rest of the adaptation process and is necessary to fine tune the prototype vectors and, therefore, provide a good quantization of the input space. During this phase the prototype vectors converge to their “correct” values. To achieve such approximation, the neighborhood should be fairly small, encompassing only the immediate neighbors. This should also apply to the learning rate, such that the magnitude of the prototype updates is very small. The convergence phase is usually several times longer than the ordering phase, e.g., at least 10 times longer (Kohonen 1996).

The above phases are illustrated in Figure 2.3 with a  $20 \times 20^4$  SOM iteratively processing 100 000 observations that describe a two-dimensional *uniform* distribution. It can be seen that a global topological ordering of the map occurs in a relatively short amount of time, e.g.,  $t = 1000$ . This is produced during the *ordering* phase, after which the network is fine-tuned slowly towards a stable state in the *convergence* phase. It is clear that the extent of the persistence in data will allow increasingly better approximations of the underlying distribution. In the provided example, after the ordering phase, at, e.g.,  $t = 5000$ , there is already a good approximation of the underlying distribution by the SOM.

#### 2.4.4.2 Parameterization

Appropriate selection of the initial learning parameters for the SOM algorithm is of critical importance to obtain topologically correct maps. Besides the random initialization

<sup>4</sup>Although rectangular SOM lattices are advised, in this illustrative case a square map allows to more easily convey the principles.



of the prototype vectors, the parameters that govern the algorithm are the learning rate  $\eta(t)$  and the width  $\sigma(t)$  of the neighborhood kernel  $h_{ck}$ . Also, the size of the lattice can influence the initial estimation of  $\sigma(t)$ . As previously discussed, the parameterization of the SOM is performed empirically, where in some cases heuristics have been proposed.

**Size of map.** The main virtue of the SOM regards the visualization of the data space, whereupon the clustering structures ought to become visible (Kohonen 2013). Because the SOM is also a vector quantization method, it has a limited resolution. To be able to detect fine structures in data, large maps, containing some hundreds of neurons, are needed (Ultsch and Siemon 1990). This is a critical necessity for the aims of this thesis. Along these larger maps and regarding heuristics, in (García and González 2004) the authors suggest setting the number of units to  $K = 5 * \sqrt{N}$ . Setting the number of neurons approximately equal to the number of input samples seems to be a useful rule-of-thumb for many applications, when data sets are relatively small (Kaski 1997). However, such heuristics can only be applied with static data.

More importantly, having a two-dimensional output layer, the lattice should take a rectangular form, rather than square, because the "elastic network" formed by the prototype vectors  $W_k$  must be oriented along with  $p(x)$  and be stabilized during the learning process. Note that if the shape were to be, e.g., circular, it would have no stable orientation in the input data space. Hence, any rectangular form is to be preferred (Kohonen 2013).

**Initialization of prototypes.** There are more sophisticated ways of initializing the prototype vectors than random initialization, but are only possible with *static* data, e.g., using input patterns or taking advantage of a PCA analysis of the input data (Kohonen 2001). The later has the effect of stretching the SOM along a direction that roughly approximates  $p(x)$ , accelerating the ordering phase. However, assuming a data stream scenario, these type of initializations are out of reach. Nevertheless, the SOM successfully unfolds over an underlying distribution, in a relatively short period of time, given the initial learning parameters are set correctly (recall Figure 2.3).

**Learning parameters  $\eta(t)$  and  $\sigma(t)$ .** Both  $\eta(t)$  and  $\sigma(t)$  should be some monotonically decreasing functions of time and their exact form is not critical; they could thus be selected linear. If the initial values for the prototype vectors are chosen randomly, both learning rate  $\eta(t)$  and neighborhood radius  $\sigma(t)$  should begin with relatively high values, thereafter decreasing gradually (Kohonen 2001); the neighborhood should be very wide in the beginning and shrink monotonically with time, until it only encompasses the adjacent neighbors. As discussed before, this is because a wide initial neighborhood kernel coupled with high learning rate, corresponding to a coarse spatial resolution in the learning process, first induces a rough global order in the prototypes, after which narrowing the kernel and decreasing the learning rate improves the spatial resolution of the map; the acquired global order, however, is not destroyed later on during the *convergence phase*.

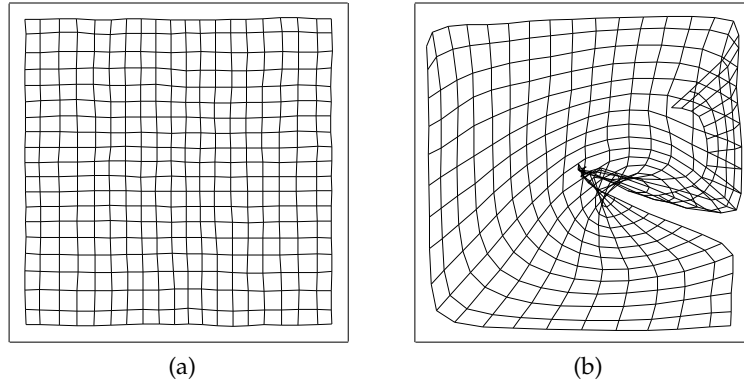


Figure 2.4: Topological ordering of the SOM when using: (a) a wide initial neighborhood kernel, and (b) a narrow initial neighborhood kernel. If the neighborhood's kernel is not wide enough in the ordering phase, then the map may exhibit topological defects.

As a general rule, neither should be decreased to zero (Kohonen 2013).

The *decreasing exponential function* is common for both  $\eta(t)$  and  $\sigma(t)$ :

$$\sigma(t) = \sigma_i \left( \frac{\sigma_f}{\sigma_i} \right)^{t/T}, \quad \eta(t) = \eta_i \left( \frac{\eta_f}{\eta_i} \right)^{t/T}, \quad (2.8)$$

where  $\sigma_i$  and  $\sigma_f$  are respectively the initial and final neighborhood width and  $\eta_i$  and  $\eta_f$  are respectively the initial and final learning rate;  $T$  is the number of iterations for which the decay is to happen. From (Kohonen 2001) it is suggested that the width of the neighborhood should be decreased from a width approximately on the order of half of the diameter of the lattice, e.g.,  $\sigma_i = 1/2 \sqrt{(width - 1)^2 + (height - 1)^2}$ , for an initial global ordering of the prototypes, down to only encompassing the adjacent neurons, e.g.,  $\sigma_f = 1$ ; the learning rate should, for example, decrease from  $\eta_i = 0.1$  to  $\eta_f = 0.01$ . Problems may arise if the initial neighborhood radius is too small to begin with, as depicted in Figure 2.4b. In this case, the network may get stuck in a “metastable state”, corresponding to a configuration with a topological defect.

#### 2.4.4.3 Limitations on Data Streams

Competitive learning limitations regarding annealing procedures for the learning parameters have already been discussed in Section 2.4.1. Nonetheless, one still has to provide  $\eta_i$  and  $\sigma_i$ . Despite no clear mathematical rules exist for establishing these values, experimentation and empirical analysis have provided some “rules-of-thumb” for setting them, as discussed above; these values normally perform well over any distribution. Also, from the above discussion, other limitations can be pointed out. The SOM indeed needs a relatively short period of time to globally order the prototypes, i.e.,  $\approx 1000$  iterations. However, considering that data streams surely span this number of iterations by some orders of magnitude, it does not seem too problematic to “sacrifice” these first observations to allow the unfolding of the map. Ways of traditionally speeding up the

unfolding rely on different strategies to initialize the prototypes, e.g., over input patterns or resorting to PCA-based linear initialization, but can only be performed having a batch of data in advance.

Regarding the size and dimensions of the map, a sufficiently large rectangular lattice, e.g.,  $20 \times 40$  (800 prototypes), should be sufficient to detect fine structures in the underlying distributions of data streams. There are alternate SOM variants that can approximate an optimal number of prototypes (see Section 2.5.2), but they all fail to maintain a proper regular topology of the map and prevent the use of visualizations for exploratory knowledge discovery (see next section).

### 2.4.5 Visualizations and Exploratory Knowledge Discovery

The currently established way of employing the SOM for knowledge discovery is through the use of large maps. This was popularized in (Ultsch and Siemon 1990), where the author called them *Emergent self-organizing maps*. From such large maps, one can perform data exploratory analysis, e.g., detect clusters of arbitrary shape and non-linear correlations between features, by using specialized visualizations. These visualizations are only possible due to the topological ordering of the prototypes, input density matching and the fixed-sized lattice of the SOM.

The SOM visualizations use colors as a visual representation of specific values — different color scales can be used. This makes them available not only to experts, but laymen, when analyzing them and can be easily understood if one is familiar with the representation. The basic visualizations<sup>5</sup> that can be derived from a SOM model are the following:

**Component Planes.** By *component plane* representation we can visualize the relative component distributions of the input data. Component plane representation can be thought as a sliced version of the SOM. Each component plane has the relative distribution of the values of one feature. In this representation, and using a temperature-like color scale, “cooler” colors represent relatively small values while “warmer” colors represent relatively large values. By comparing component planes we can see if two components correlate. If the outlook is similar, the corresponding features correlate; if they seem like the “negative” of each other, then the corresponding features are inversely correlated.

**Unified Distance Matrix.** The *unified distance matrix*, or simply *U-Matrix*, presents distances between neurons. The distances between adjacent neurons are calculated and presented with different colorings between the adjacent neurons (prototypes). Following the same temperature-like color scale, a warmer coloring between neurons corresponds to a large distance and thus a gap between the codebook values

<sup>5</sup>A third basic visualization concerns the *Hit Histogram* which shows how many observations are mapped onto each SOM neuron. Since it cannot be derived directly in a stream setting, it was purposely left out of this list. A similar visualization is later introduced in Chapter 7.

in the projected space; a cooler coloring means codebook vectors are close to each other in the input space. Cooler areas can be thought as clusters and warmer areas as cluster separators. This is specially powerful to understand the underlying structure of data, i.e., when one tries to find clusters in the input data without having any a priori information about the clusters. Consequently, the detection of complex clusters is achieved not by regarding single units, but by regarding the topological structure map.

Hence, while component planes allow discovery of relationships among features, the U-Matrix allows the visual discovery of clusters of data. Used in conjunction, one may infer which features best describe the detected clusters, i.e., a description of the clusters. It must be emphasized that this knowledge discovery through visual exploration is the main motivating factor for using SOMs over data streams, regarding current available methods.

A simple illustrative example of these visualizations is provided in Figure 2.6 for the well-known Fisher’s *Iris* dataset (Lichman 2013), obtained from a  $20 \times 40$  SOM. In summary, the dataset contains 150 observations from three species of flowers, namely *Iris setosa*, *Iris versicolor* and *Iris virginica*, regarding measurements of *petal* and *sepal width* and *length* in centimeters, hence  $d = 4$ ; both observations and SOM lattice are depicted in three-dimensions using PCA projection (Section 2.4.3). It is well-known that one cluster (containing *Iris setosa* flowers) is linearly separable from the other two. From the U-Matrix these two clear clusters can be derived. Note that in this example the dataset consists in only 150 observations, while the SOM contains 800 prototypes; for this reason, the projection is very detailed and several micro structures seem to be contained inside each detected cluster. Also, the input space density matching of the SOM gives an idea of relative cluster sizes, e.g, the top cluster is approximately half the size of the bottom cluster. Indeed, the dataset contains 50 observations of each flower. Through component planes we observe that values for *petal length* and *petal width* are distributed in a similar manner and, consequently, these features are correlated. By comparison with the U-Matrix we detect that the top cluster is composed of flowers that have smaller petals, which is actually the description of the *Iris setosa* flowers and what makes this cluster linearly separable.

Another aspect depicted in this example regards the non-optimal quantization performed by the SOM, where several “unrepresentative” prototypes can be seen. Although they do not represent actual observations they can be seen as *interpolators* in the sense of data generalization. Nonetheless, the SOM concentrates much more, and closer, prototypes in the denser input areas, which is ultimately captured in the U-Matrix.

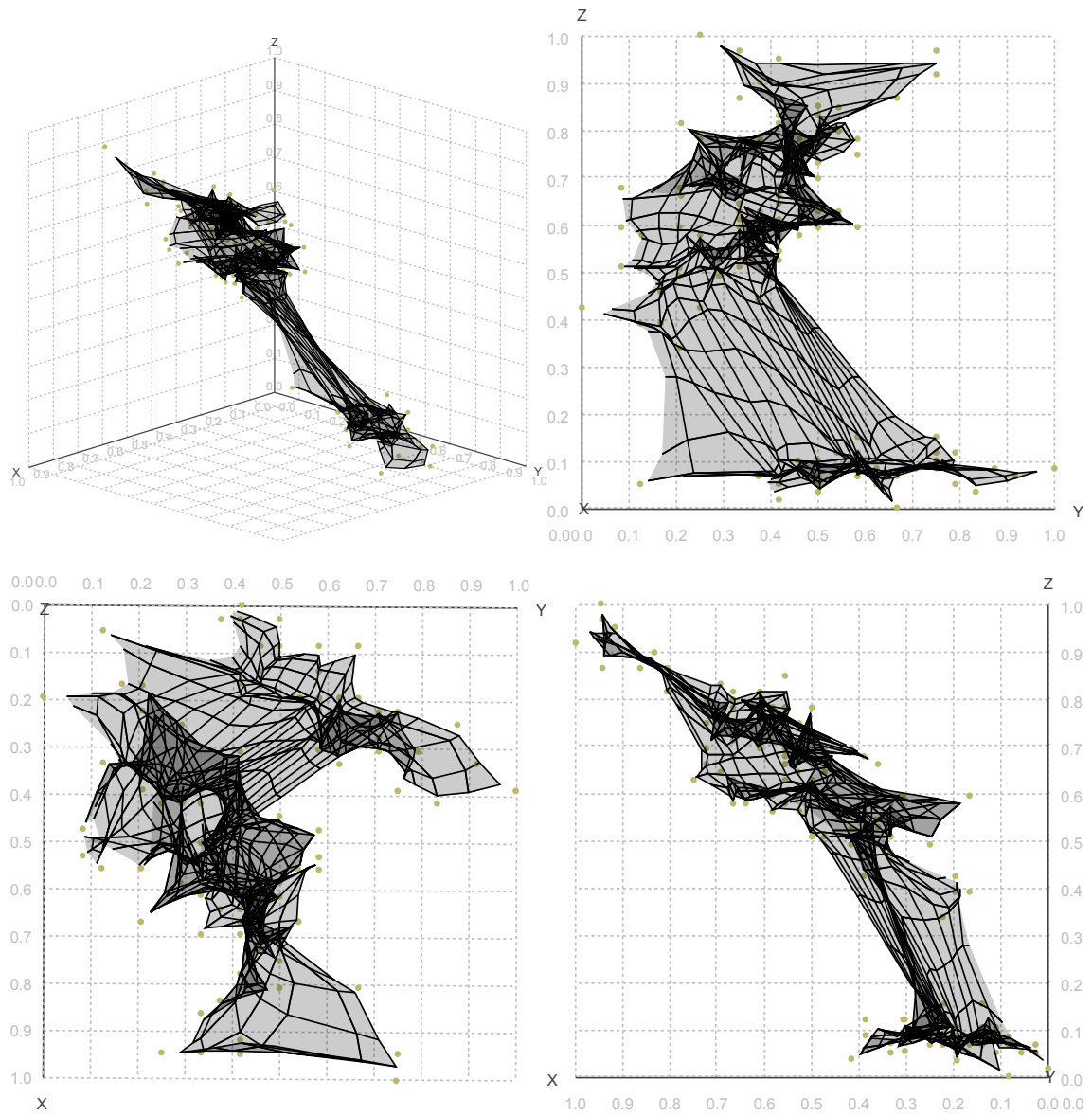


Figure 2.5: Depiction of a  $20 \times 40$  SOM quantizing the IRIS dataset from different visual perspectives (PCA projection was used to obtain a 3D visualization).

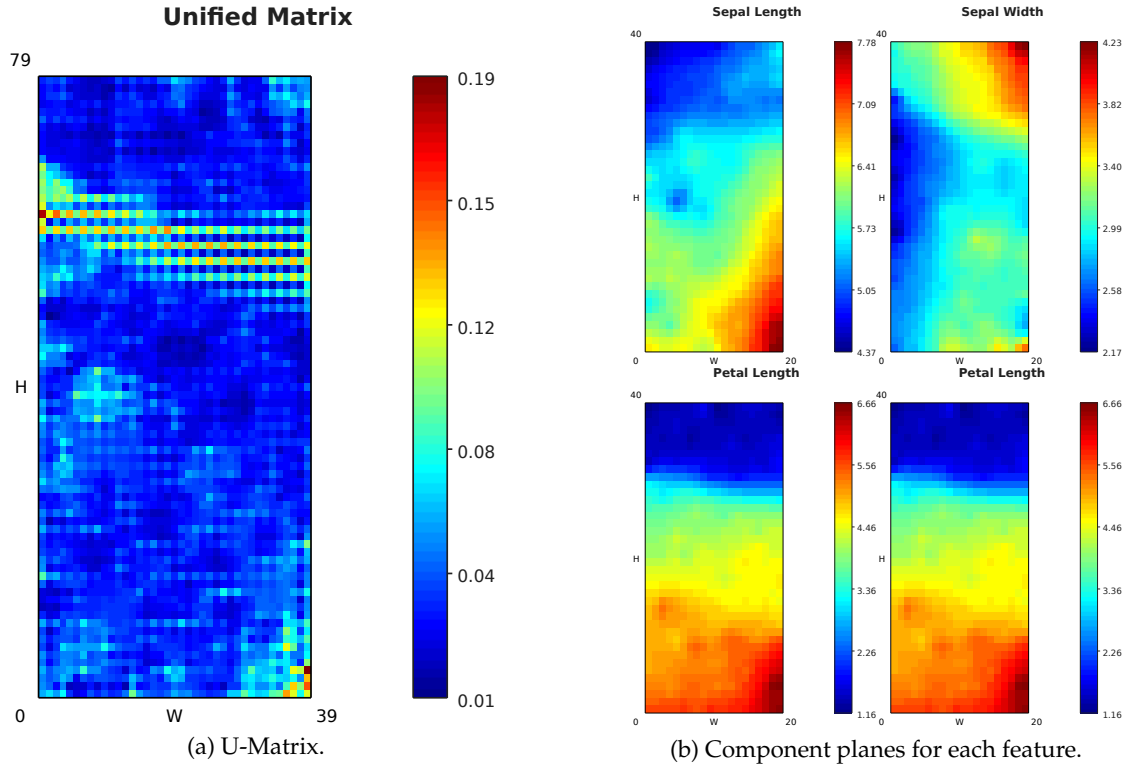


Figure 2.6: Examples of visualizations derived from a  $20 \times 40$  SOM after learning the Iris dataset.

### 2.4.6 Quality Assessment

In the context of static data, although some optimal map exists for a particular dataset, depending on the parameterization, different maps arise. Consequently, it is important to know whether the map has properly adapted itself to the input data. Within many other methods the evaluation is usually performed against a cost function that defines the optimal solution, e.g. the MSE. However, as mentioned in Section 2.4.4, no cost function has been derived for the SOM. Hence, other quality measures are used. Measuring the quality of obtained maps can be done in several ways, but all require the presence of the entire dataset. Two of the most important properties these measures usually try to evaluate are the somewhat conflicting goals of VQ and VP. This conflict is especially obvious when the dimension of the input data is higher than the dimension of the output network (Kirk and Zurada 1999). Being both a VQ and VP method, the most simple and used quality measures regarding these goals are the *mean quantization error* (QE) and *topographic error* (TE) (Kohonen 2001). While measuring the quality of an obtained SOM, consideration must be taken for both these quality measures.

### 2.4.6.1 Mean Quantization Error

The mean quantization error  $\overline{QE}$  is related to all forms of vector quantization and competitive learning clustering algorithms, disregarding map topology. The mean QE measures how well the map quantizes the input data and the best map is expected to yield the smallest mean quantization error between  $\mathbf{x}(t)$  and corresponding  $\mathbf{w}_c(t)$ . It is computed by determining the mean distance of input patterns to the prototype vectors of the BMU's that represent them, i.e., mean of the "local" quantization errors  $E_q(t)$ :

$$\overline{QE} = \frac{1}{N} \sum_{t=0}^{t=N-1} E_q(t) \quad (2.9)$$

$$E_q(t) = \|\mathbf{x}(t) - \mathbf{w}_c(t)\| \quad (2.10)$$

where  $N$  is the number of input patterns. A SOM with a lower  $\overline{QE}$  is more accurate than a SOM with higher mean error.

### 2.4.6.2 Mean Topographic Error

The mean topographic error  $\overline{TE}$  measures how well the topology is preserved by the map. Unlike the mean quantization error, it considers the structure of the map. It is the most simple measure for topology preservation. For all input patterns, the respective BMU and the second-BMU are determined. If these are not adjacent on the map lattice, this is considered an error. Let  $\mathbf{w}_c(t)$  and  $\mathbf{w}_{c'}(t)$  be the BMU and second-BMU, respectively, for the observation  $\mathbf{x}(t)$ . Then the  $\overline{TE}$  is given by the mean of all local topographic errors  $E_t(t)$ :

$$\overline{TE} = \frac{1}{N} \sum_{t=0}^{t=N-1} E_t(t) \quad (2.11)$$

$$E_t(t) = \begin{cases} 0 & \text{if } adjacent(\mathbf{w}_c(t), \mathbf{w}_{c'}(t)) \\ 1 & \text{otherwise} \end{cases} \quad (2.12)$$

The  $\overline{TE}$  can be seen, basically, as a measure of continuity of the mapping, where low values indicate a smooth mapping, with similar observations mapped to close-by units. The topographic error aims at detecting cases where topological defects occur, such as the one illustrated in Figure 2.4b. However, even with perfect topology preservation, the topological error may not be zero due to the way the SOM covers the input space — this will be specially obvious when presenting examples of the SOM mapping a single Gaussian distribution, e.g., Figures 4.14a and 5.14.



### 2.4.6.3 Other Quality Measures

Several other quality measures exist, such as the *topographic product* (Bauer and Pawelzik 1992), *distortion* (Vesanto, Sulkava, and Hollmen 2003) and *trustworthiness* and *neighborhood preservation* (Venna and Kaski 2001). The topographic product can be used to assess whether the size of the map is appropriate to fit onto the dataset, while the SOM distortion measure can be used to select the best fitting map from several trained maps with the same dataset. Trustworthiness and neighborhood preservation measures are similar and provide a value that indicates if vectors close in the input space are mapped to neighboring neurons in the map. Further discussion and comparison of these measures can be found in (Polzlbauer 2004).

Overall, application of any of the above quality measures over data streams implies their adaptation to unbounded data.

### 2.4.7 Comparison of Clustering Methods

A plethora of clustering methods have been developed over the years, mainly because the notion of similarity/dissimilarity, and consequently “cluster”, is not precisely defined. Therefore, different clustering methods may generate different clusterings on the same data. Authors in (Han, Kamber, and Pei 2006) suggest categorizing clustering methods into five main categories: *partitioning*, *hierarchical*, *density-based*, *grid-based* and *model-based*. Some proposed methods may fit into more than one category. While based on competitive learning and therefore having a partitioning aspect, the SOM is more sophisticated because the topology preservation property enables it to provide a compact model of the data from where, through visualizations (Section 2.4.5), insight on the data structure can be obtained without prior assumptions (Ultsch 1995). Hence, the SOM can be characterized as a model-based method in this taxonomy of methods.

A brief summary of the most well-known methods that fit into the above taxonomy is provided with two aims: (i) to provide a clear comparison of the SOM clustering results regarding other methods, and; (ii) because clustering methods operating on data streams generally use adaptations of some of these methods, as will be surveyed in Section 2.6. Together, they should provide a clear motivation for the use of self-organizing maps in exploratory cluster analysis over data streams. The following subsections focus on the problem of *clustering observations*, while *feature clustering* is addressed separately in Section 2.4.7.6.

Examples of the type of clustering results provided by the compared methods are illustrated in Figure 2.7. using the same data. Each of the individual results were obtained through the ELKI framework (Achtert et al. 2012), where a SOM implementation was contributed from this research, and are referenced in the following subsections. The *Density* dataset (Figure 2.7a) is already in normalized form, i.e.,  $\mathbf{x} \in [0, 1]^2$ , and is provided



with the framework. The described distribution is composed by two different shaped clusters (100 and 270 points marked as + and ×, respectively) with similar densities and containing outliers (20 points marked with ○), totaling 390 observations.

### 2.4.7.1 Partitioning Methods

Partitioning methods relocate objects by moving them from one cluster to another, starting from an initial partitioning. Such methods typically require that the number of clusters  $K$  will be defined *a priori*. The global optimal solution is a NP-hard problem, therefore greedy heuristics are used in the form of iterative optimization. The simplest and most commonly used algorithm, employing a MSE criteria, is the *k-means* algorithm. This algorithm partitions the data into  $K$  clusters  $\{C_1, C_2, \dots, C_K\}$ , represented by their centers or means (centroids). The k-means algorithm begins with an initial set of  $K$  cluster-centers and updates it so as to decrease the error criterion. The incremental k-means — whose relation with competitive learning was addressed in Section 2.4.1.1, is suitable for a training set that is obtained on-line. In the batch k-means at each step the patterns keep changing from one cluster to the closest cluster  $C_j$ , according to the nearest-neighbor rule, and the prototypes are then recalculated as the mean of the samples in the clusters during a predefined number of iterations (epochs in competitive learning terminology). K-means is susceptible to local *minima* of its objective function, depending on the initialization of the centers  $C_j$ . Also, selecting the appropriate value of  $k$  is a difficult task without a prior knowledge of the input data. Also, they can be sensitive to noise and outliers and are based on the assumption that clusters are spherical and of equal density. A variation, more robust to noise and outliers, is the *k-medoids* method (Kaufman and Rousseeuw 1987). This algorithm is very similar to the k-means algorithm, but each cluster is represented by the most centric observation of a cluster, i.e., the *medoid*, rather than by the implicit mean that may not belong to the cluster. Batch k-means has a time complexity of  $O(NKdT)$  and space complexity of  $O(Kd)$ , where  $T$  is the number of iterations. This linear complexity is one of the reasons for the popularity of the k-means algorithm. Even if the number of instances is substantially large, this algorithm is computationally attractive.

Figure 2.7b illustrates the resulting k-means clustering over the *Density* dataset, setting  $K = 2$ . This dataset clearly depicts the deficiencies of the algorithm in detecting the natural clusters by assuming spherical clusters; the outliers also affect the centroid positions.

The clustering performed by ART networks (described in Section 2.4.1.2) falls into partitioning methods, although realizing a variable number  $K$  of prototypes, dependent on the vigilance parameter  $\rho$ . Time complexity is of  $O(NKd)$  and space complexity of  $O(Kd)$ , but with varying  $K$ . Figure 2.7c illustrates the clustering performed by the ART2-A algorithm, setting  $\rho = 0.95$ , and the resulting *perceptive fields*. This illustrates the critical

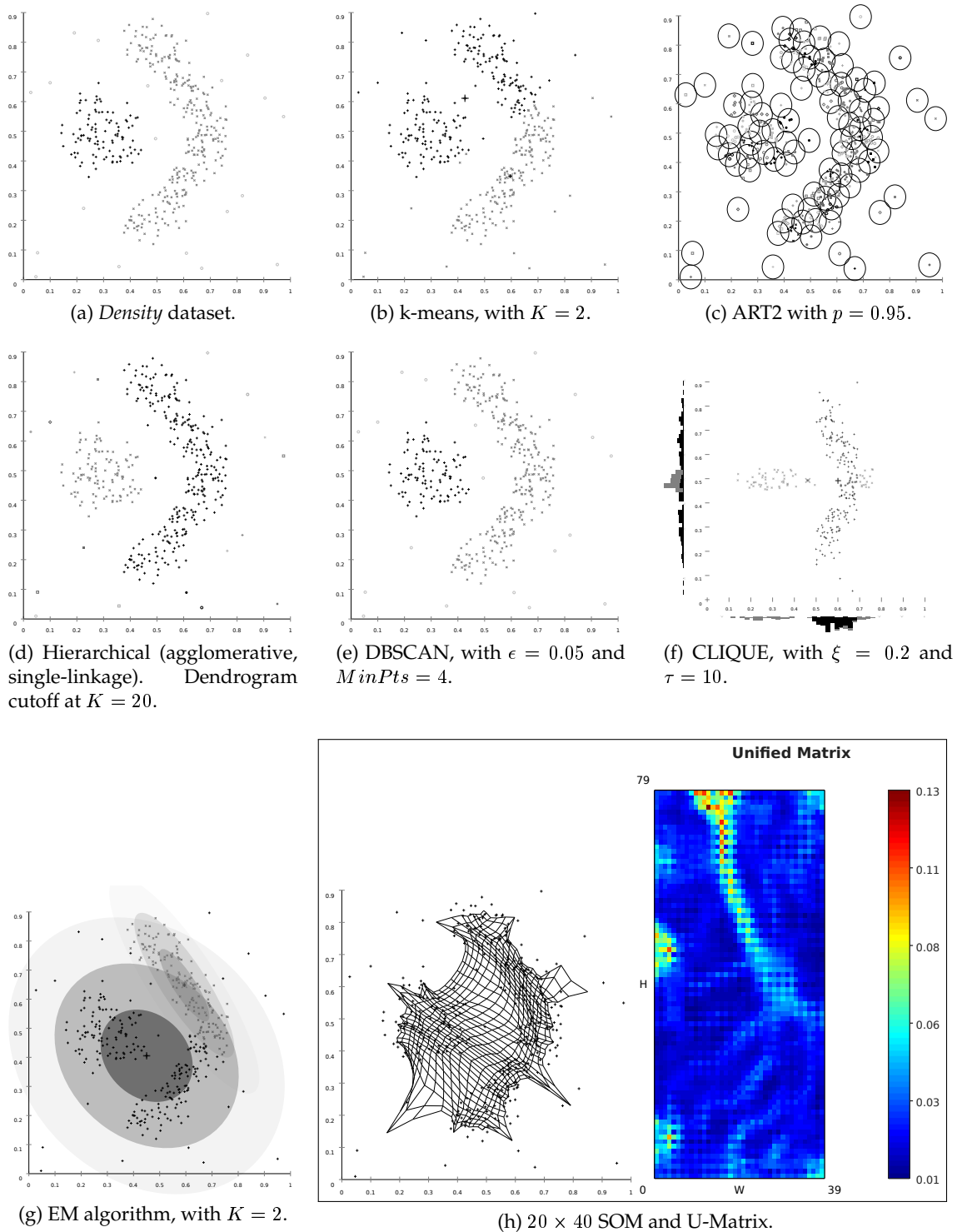


Figure 2.7: Comparison of clustering results by different algorithms.

impact of the vigilance parameter and the fact that ART2-A networks are not robust to outliers. If we were to adjust the vigilance parameter  $\rho$  to obtain exactly 2 clusters, the results could be similar to k-means, but would heavily depend on the order of presented observations. A more recent variation of the ART2-A algorithm, called *Regional and On-line Learnable Fields* (ROLF) (Schatten, Goerke, and Eckmiller 2005), establishes an individual width parameter  $\sigma$  to each individual category, describing the variance of points assigned to them. The suggested clustering result relies on sets of overlapping perceptive fields, defined by their radius ( $r = \rho \times \sigma$ ), to establish the natural clusters; it would be successful in detecting the two underlying clusters in the illustrative example.

#### 2.4.7.2 Hierarchical Methods

Hierarchical clustering is a nested sequence of partitions. A clustering result is created by building a tree (also called a *dendrogram*) either from the leaves to the root (*agglomerative* approach) or from the root down to the leaves (*divisive* approach) by merging or dividing clusters at each step. In order to decide which clusters should be combined (agglomerative) or split (divisive), any distance measure can be used and the linkage criterion determines the distance between sets of existing clusters as a function of the pairwise distances between observations included in the clusters. Here *single*-, *complete*- or *average-linkage* criteria can be used, which define the cluster distances based on the minimum, maximum or average distance between observations in any two clusters, respectively. Outliers may be easily identified in hierarchical clustering, since they, e.g., merge much later in the clustering process.

A partitioning clustering of the data can be obtained by cutting the dendrogram at a desired similarity level or, more commonly, such that the cut traverses exactly  $K$  branches of the dendrogram. The number of clusters  $K$  need not be specified in advance and the local minimum problem arising from initialization does not occur. Also, it can detect clusters with arbitrary shapes. However, hierarchical clustering is static, and observations committed to a given cluster cannot move to a different cluster. Also, the linkage criteria deeply affects how the clusters are merged or split.

Building a tree using either an agglomerative or a divisive approach can be prohibitively expensive for large datasets. Hierarchical methods require  $O(N^2)$  space for the similarity matrix. Time complexity is  $O(N^2d)$  for distance computation and  $O(N^3d)$  for complete clustering procedure.

Figure 2.7d depicts the resulting clustering obtained from an agglomerative procedure with single linkage criterion by cutting the dendrogram to obtain 20 clusters. This was the cutoff point that effectively matched the two natural clusters, while the remaining 18 clusters contain single noise observations. Although being able to detect arbitrarily shaped clusters, determining the correct cutoff point is not an easy task.

### 2.4.7.3 Density-based Methods

Density-based methods group observations into clusters based on density conditions: clusters are dense regions of observations in the data space and are separated by regions of low density. Their general idea is to continue growing a given cluster as long as the density (number of observations) in the “neighborhood” exceeds some threshold. For example, for each observation within a given cluster, the neighborhood of a given radius has to contain at least a minimum number of other observations to effectively be considered a cluster; otherwise they are considered as noise. Such methods can be used to filter out noise or outliers and discover clusters of arbitrary shape.

One example is the *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) method (Ester et al. 1996). DBSCAN features a well-defined cluster model called “density-reachability”. It is based on connecting objects (observations) within certain distance thresholds. However, it only connects objects that satisfy a density criterion — in the original variant defined as a minimum number of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of arbitrary shape) plus all objects that are within these objects range. DBSCAN needs two global parameters, namely  $\epsilon$  and  $MinPts$ , which determine the threshold for density-reachability and minimum number of objects to generate a cluster, respectively. Its time complexity is  $O(N(\log N)d)$  and space complexity is  $O(Nd)$ . Since it needs access to the entire dataset, by itself alone it is not applicable to data streams.

Figure 2.7e illustrates the DBSCAN algorithm, with  $\epsilon = 0.05$  and  $MinPts = 4$ . It is successful in detecting both clusters and identifying the outliers (because of the point threshold count). However, the quality of the results depend heavily on the correct choice of parameters that match the density of the clusters. With clusters of different densities, the DBSCAN algorithm produces inferior results, given the fixed set of parameters assumes similar densities.

### 2.4.7.4 Grid-based Methods

Grid-based methods discretize the input space into a finite number of non-overlapping cells (hyper-rectangles), i.e., the “grid”, and then perform the required clustering operations on the discretized space. The main advantage of grid-based method is its fast processing time which depends on number of cells in each dimension in discretized space (Ilango and Mohan 2010). Therefore, grid-based methods are often integrated with other clustering methods such as density-based methods and hierarchical methods. However, selecting the appropriate size for the grid may not be straightforward.

One example of an algorithm that makes simultaneous use of concepts of grid and density-based methods is the *Clustering In Quest* (CLIQUE) algorithm (Agrawal et al. 1998). Its purpose is to automatically find subspaces within high-density clusters in high

dimensional data. The basic idea is that if a collection of objects  $P$  is a cluster in a  $d$ -dimensional space, then  $P$  is also part of a cluster in any  $(d - 1)$ -dimensional projections of this space. The grid is obtained by partitioning every dimension into intervals of length  $\xi$ . A cell is considered dense if the total number of points contained in it is more than the threshold  $\tau$ . By comparing the density of points in individual dimensions, subspace clusters can be identified as well as a minimal description for the clusters. Compared to DBSCAN it is easy to observe that the density-reachability criteria is replaced by adjacency of grid cells.

Figure 2.7f presents an illustrative result of the CLIQUE algorithm, with  $\xi = 0.2$  and  $\tau = 10$ , where the individual counts of objects assigned to each cell dimension-wise are visible. The algorithm finds dense clusters in the subspaces (individual dimensions), that can be later compared to find overlapping areas in any arbitrary two or more dimensions. The established cell size and minimum number of points filters the outliers, but also some cluster objects.

#### 2.4.7.5 The SOM and Other Model-based Methods

These methods attempt to optimize the fit between the given data and some mathematical model. Besides allowing the detection of clusters, model-based clustering methods may also find characteristic descriptions for each cluster. Some of the most frequently used methods are decision trees, *Gaussian Mixture Models* (GMM) and neural networks, being the SOM the more predominant method of the later.

Decision trees are mostly well-know for their use in classification tasks, but they have been developments for their use in unsupervised learning (Liu, Xia, and Yu 2000). GMMs are probability distributions used in clustering and density estimation, and a subset of *finite mixture models* (Melnykov and Maitra 2010). A GMM assumes the PDF is composed by  $K$  *Gaussians*, by maximizing data likelihood. While k-means can only find spherical clusters, a GMM is able to refine such clusters to ellipsoid shapes. Maximum likelihood estimation is possible via the Expectation-Maximization (EM) algorithm (McLachlan and Krishnan 2007). It follows an iterative, sub-optimal, approach which tries to find the parameters of the probability distributions, e.g., the mean and co-variance Gaussian parameters, that has the maximum likelihood of its attributes. The main disadvantages are the same as for k-means, i.e., assumption of regular shaped clusters and providing the desired number of clusters  $K$ . Figure 2.7g illustrates the result of a GMM obtained via the EM algorithm, setting  $K = 2$ . In this case, the established GMM fail to detect the natural clusters, but, given the shape of the cluster on the right, this was expected. By establishing  $K = 3$ , this cluster is effectively represented by two Gaussians. The EM algorithm does not predict the existence of outliers and noise.

A more or less extensive description of the SOM has already been given, reinforcing

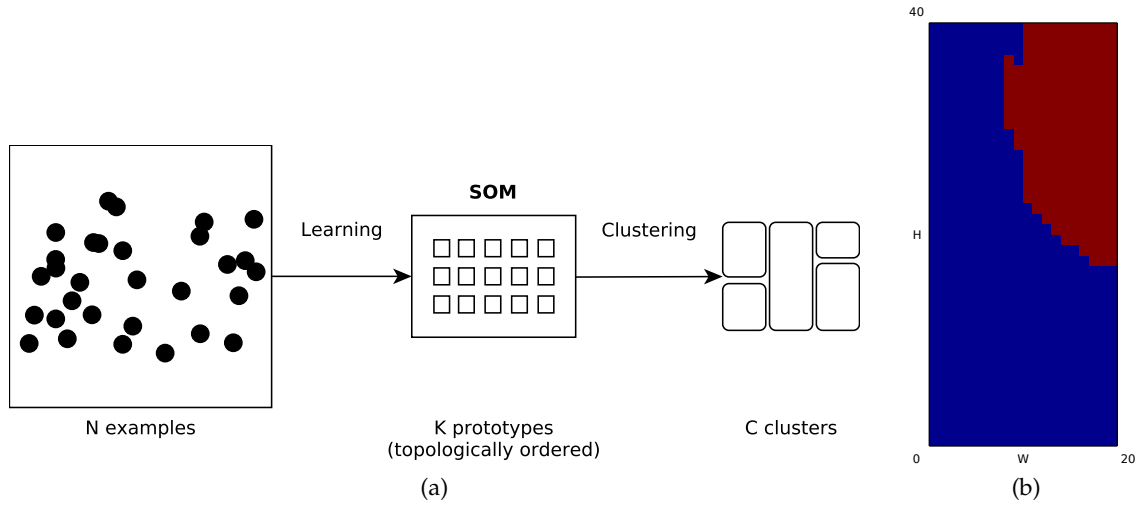


Figure 2.8: Two-stage clustering of the SOM: (a) First abstraction level is obtained by creating a set of prototype vectors using the SOM. Clustering of the SOM creates the second abstraction level; (b) Example of SOM-Ward method over SOM lattice of Figure 2.7h.

its potential in exploratory cluster analysis, not only by visual inspection of the underlying cluster structure in data by means of the *U-Matrix*, but by also providing a description of those clusters via component planes visualizations (see Section 2.4.5). Figure 2.7h illustrates a  $20 \times 40$  SOM lattice obtained after learning the *Density* dataset, with the parameters empirically set in Section 2.4.4, using 10 ordering epochs and 40 convergence epochs. The corresponding *U-Matrix* is also depicted, where two clear clusters are easily seen. The robustness to the outliers is somewhat satisfactory. What remains relevant is the fact that no assumptions were needed regarding the desired number of clusters; the SOM application enabled the detection of two clusters with arbitrary shape.

### Clustering of the SOM

Although not applied in this thesis, if the goal is to create a few, but quantitative clusters, the SOM has to be clustered. In such situations, a two-level approach was proposed (Vesanto and Alhoniemi 2000), where a dataset is used to obtain a SOM model, which is then clustered through other methods. This concept is illustrated in Figure 2.8a. However, the detection of complex clusters is achieved not by regarding single prototypes, but by regarding the topological structure map, using methods such as the *SOM-Ward* (Viscovery SOMine 6.0) and *Lake* (Matos, Marques, and Cardoso 2014) methods. For example, the former method employs an agglomerative hierarchical algorithm with the *Ward* criteria (Ward Jr 1963), but attending to the topology of the SOM by assigning infinite distances between prototypes that are not adjacent in the lattice. An example of the *SOM-Ward* method applied to the previous SOM over the *Density* dataset is depicted in Figure 2.8b, by establishing the cutoff on the *Ward* dendrogram such that  $K = 2$ . Then



each prototype is labeled with the specific color. This can aid visual inference on the U-Matrix, but more importantly allows a partitioning of data following a classification task approach: for each observation, find the BMU and assigning the corresponding label to the observation. Several studies have shown the effectiveness of this approach over some classical clustering algorithms, namely k-means (Lee, Gu, and Suh 2006; Samarasinghe 2006; Vesanto and Alhoniemi 2000). However, the SOM-Ward method is not effective in detecting all types of clusters. An example is the cluster structure described in the *Complex* data stream used later in this thesis (Figure B.1b); it cannot detect clusters within clusters.

#### 2.4.7.6 Clustering Features

The task of clustering features can have different aims, such as dimensionality reduction and corresponding feature selection or simply detecting correlated features (Dash and Liu 2000). The later can be a simple process of knowledge discovery to comprehend data. For the objective of dimensionality reduction, PCA is a popular dimension reduction technique, but its interpretation of input space is not as good. An alternative is to group strongly correlated features and perform selection of representative features in all groups, effectively performing dimensionality reduction (James et al. 2013). A simple and frequently used approach for clustering a set of features is transpose the dataset and apply a traditional clustering algorithm, such as the ones presented in the previous section. More usually, hierarchical methods are employed over a dissimilarity matrix. The dissimilarities between data can be computed either with Euclidean distances or correlation coefficients, e.g., *Pearson* correlation coefficient. The later allows the discovery of either directly and inversely correlated features. Such example is the VARCLUS<sup>6</sup> procedure from the *R* language, which has different similarity metrics to choose from.

While discussing the SOM visualizations in Section 2.4.5, the component planes were presented as a means to give a description of the detected clusters on the U-Matrix, but also as a means to detect correlated features. Indeed, in SOM literature this is referred to as *correlation hunting* (Vesanto and Ahola 1999). By transforming component plane values into vectors, e.g., row- or column-wise, clustering of the features can be indirectly performed. Works such as (Pérez-Urbe 2007; Vesanto and Ahola 1999) use this reasoning to reorder the presentation of the component planes, allowing the user to more easily detect correlated features when their number is very high. Both works achieve this by clustering the component planes with another SOM, varying in the way they are presented to the user.

As mentioned previously, clustering features is strongly related to time-series clustering, where the objective is to find time-series that somehow behave similarly along time or are correlated. Time series clustering has been shown effective in providing useful

<sup>6</sup><http://www.inside-r.org/packages/cran/hmisc/docs/varclus>

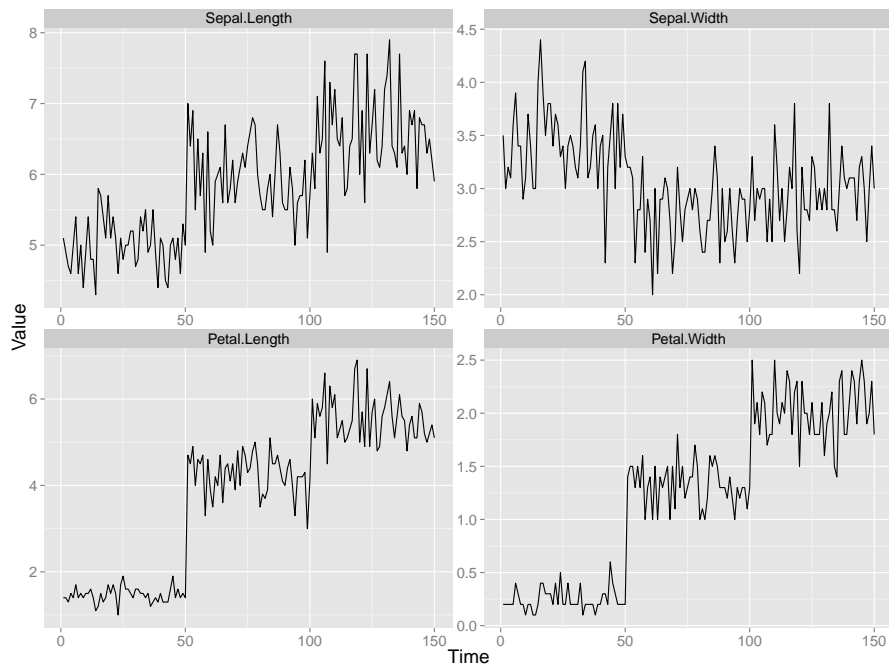


Figure 2.9: Iris dataset represented as a data stream.

information in various domains. There seems to be an increased interest in time series clustering as part of the effort in temporal data mining research. A lot of research effort has been put into this task, either considering time-series of equal or different lengths as subsequences, with raw-based methods (that apply similar strategies to feature clustering), feature-based or model-based methods. However, clustering subsequences in time-series has generated some controversy in literature (Keogh and Lin 2005). An in-depth overview of these methods can be found in (Liao 2005).

An indication that the SOM can be successful in such tasks is exemplified by plotting the previous *Iris* dataset as a multivariate time-series, see Figure 2.9, where each type of flower is plotted sequentially. If we recall Figure 2.6b with the corresponding SOM component planes, *petal length* and *petal width* (bottom time-series in Figure 2.9) were found to be correlated. As such, we can extrapolate that such methodology can be applied to time-series clustering (of equal length). Based on this assumption and correlation hunting in general, this research proposed a general feature clustering method for SOM's component planes, based on hierarchical clustering, which was applied successfully to financial time-series for the purpose of portfolio selection (Silva and Marques 2010a).

## 2.5 Evaluation of the SOM and Related Variants for Data Streams

From the previous comparison between clustering methods, it should be clear that the SOM excels in data visualization and exploratory cluster analysis, consequence of the topological models from where simple visualizations highlight the data structure. However, as discussed, the original SOM is not suited for unbounded and non-stationary data



streams due to the annealing schemes used in determining the learning parameters evolution, i.e., they are time-dependent and as time passes the model loses plasticity. In this section we are interested in determining requirements SOM variants should possess regarding data streams and evaluate if existing variants fit some or all of them.

### 2.5.1 Proposed Requirements

In Section 2.3 the characteristics clustering algorithms should possess regarding data streams were enumerated. Consequently, these are desirable requirements we should have in mind. Furthermore, the SOM derives all of its usefulness in exploratory cluster analysis from its visualizations, which rely on a fixed and regular output layer (the rectangular lattice). Also, the time-dependent learning parameter estimation is a serious deterrent aspect on its applicability to data streams. On the other hand, the annealing schemes are critical for proper convergence of the maps (Section 2.4.4). As such, the requirements SOM variants should address regarding data streams are established as follows:

- i. Fixed topology, i.e., a regular lattice of neurons to allow visualization procedures;
- ii. Time-independent learning parameter estimation;
- iii. Proper convergence to the underlying distribution in the sense of a VQ procedure;
- iv. Incremental and efficient processing of individual observations;
- v. Compactness of model;
- vi. Dealing with evolving data (non-stationarity);
- vii. Robustness to noise, and;
- viii. Handling high-dimensional data.

The first three enumerated properties are SOM-related, while the remaining are general to any data stream clustering algorithm (see Section 2.3.1).

### 2.5.2 Variants of the SOM

From its inception the original SOM algorithm has led to the proposal of dozens of related algorithms and variants aiming at different particular purposes and applications. In (Baao, Lobo, and Painho 2008) three main areas in which the original SOM can be modified are identified:

- Topology and connection between neurons;
- Matching mechanism, i.e., determining the BMU, and;
- Learning rule, i.e., prototype update mechanism;

Variants of the SOM that change the matching mechanism are usually variants that target heterogeneous types of features, e.g., including spatial data (Baç o, Lobo, and Painho 2005b) and categorical data (Chen and Marques 2005), usually also changing the update rule to take into account these different types of features. Another example is the use of “conscience” mechanisms (DeSieno 1988) to reduce the number of unrepresentative prototypes in the SOM codebook, by attaching counters to each neuron so as to enable each neuron to have the same opportunity (probability) to represent an observation.

Despite the huge amount of literature around SOM and SOM-like networks, there is surprisingly and comparatively very little work dealing with incremental learning. Incremental learning addresses the ability of repeatedly training a network using new data, without destroying old prototype patterns (Furao, Ogura, and Hasegawa 2007). This is somewhat related to evolving and non-stationary data, but may not predict forgetting old information. Most of these works alter the structure of the network, creating and deleting nodes as necessary. There are also variants that use (possibly growing) hierarchies of SOM, e.g., (Rauber, Merkl, and Dittenbach 2002), to represent parts of the input space with different granularities, but they are out of scope for this research. Given fixed topology is one of the main concerns for data visualization, surveyed proposals will be distinguished between models that vary the topology of the lattice and others that keep it fixed.

### 2.5.2.1 Varying Topology

Proposals that vary the topology of the lattice are able to deal more easily with evolving data by adding and removing neurons from the network. This may have impact on the compactness of the model requirement, but more importantly, hinder or make data visualization very difficult. They indirectly ensure convergence, i.e., proper VQ, by inserting neurons in denser areas of the input space.

### Growing Cell Structures

The Growing Cell Structures (GCS) model (Fritzke 1994) allows nodes to be added and removed dynamically, but without imposing obligatory neighboring relations, i.e., a node may not have connections with others. The model starts with a predefined number of nodes in a specified arrangement, i.e., neurons can have an arbitrary number of neighbors. The rationale is to represent increasing complex regions of the input space with denser lattices. The growing process is based on a counter value attached to each node, which is incremented by one every time the neuron is identified as the BMU. Upon reaching a threshold value, the corresponding node is considered to be a good place to insert a new node into the network. The neighbor with the largest distance to this node is selected and a new node is inserted between these two, defining the respective neighboring relationships. The principle behind this idea is that if cells match very well over a long

period of time, then they must be in a region of dense input patterns where more nodes are needed. All counters are decreased within each time step by a small fraction in order to keep track of time. The same principle applies to cell removals. As soon as a counter value falls below another threshold, the corresponding cell is regarded as laying in a region of low density probability and is therefore removed from the network. The input space mapping may result in several separate regions of neurons with varying number of neighboring relations. It uses constant learning rates. The same author, shortly after, proposed the *Growing Neural Gas* (GNG) algorithm (Fritzke 1995), which shares a lot of similarities.

### **Evolving Self-Organizing Map**

The *Evolving Self-Organizing Map* (ESOM) (Deng and Kasabov 2003) is based on an incremental network quite similar to GNG that creates nodes dynamically based on the distance measurement between the BMU and the current observation, but the new node is created at the exact data point instead of the mid-point as in GNG. The ESOM network starts without nodes. During learning, the network updates itself from sequential observations, creating new nodes when necessary. Nodes are created if no suitable node is found to represent an observation, based on a distance threshold  $\epsilon$ , i.e., if  $E_q(t) > \epsilon$ . It uses a constant learning rate, otherwise. Connections between nodes are used to maintain the neighborhood relationships between close nodes. The strength of the neighborhood relation is determined by the distance between connected nodes. If the distance is too big, giving a weak strength under a threshold, the connection can be pruned. Deriving from connection pruning, isolated neurons are also deleted. In this way the feature map can be split apart and data structures such as clusters and outliers can emerge (noise is considered outlier data). However, no implicit or explicit forgetting mechanism exists. By removing only isolated neurons, the network is unable to forget a portion of the input space that may have become obsolete.

### **Enhanced Self-Organizing Incremental Neural Network**

*Self-Organizing Incremental Neural Network* (SOINN) (Furao and Hasegawa 2006) and its *Enhanced* version (ESOINN) (Furao, Ogura, and Hasegawa 2007) are also based on an incremental structure, where the former version uses a two-layer network and the later a single layer network. The ESOINN algorithm starts with two randomly initialized prototypes (neurons) and is very similar to GCS. New nodes are always inserted between two existing nodes, uses edge aging to remove neighboring relations and forgets old information based on counter values attached to each neuron. The insertion criteria is different, leading to triangle-mesh like mappings of the input space. It also uses an annealing scheme for the learning rate.

### 2.5.2.2 Fixed Topology

Some SOM variants that preserve topology, keep the size of the map fixed and use time-independent learning parameters have been more recently proposed. By keeping a fixed topology and allowing the application of the visualization procedures, these variants are of particular interest and will be presented in more detail.

The basic idea behind the following proposals is that for an input pattern that the network already represents well, there is no need for large adjustments — learning rate and neighborhood radius are kept small. On the other hand, if an input pattern is very dissimilar of what was previously seen, then those parameters are adjusted to produce large adjustments. However, both proposals use the local error  $E_q(t)$  to estimate learning parameters, which makes them very sensible to noise. Also, despite inducing an indefinite *plasticity*, they are unable to properly converge to a density mapping VQ — recall that monotonically decreasing learning parameters are needed to ensure this convergence (Kohonen 2001). In summary, what is actually mapped by these proposals is the structure or support of the distribution rather than the density.

### Parameter-Less SOM

The Parameter-less Self-Organizing Map (PLSOM) (Berglund 2010) was initially proposed as a way to reduce the parameter-space of the original SOM algorithm. It has only one constant parameter  $\gamma$  (neighborhood range) that needs to be specified, after which learning parameters are estimated dynamically at each iteration. In the PLSOM the amplitude and range of the prototype updates are not dependent on time  $t$ , but on how well the map fits the input data. The “fit”  $d(t) \in [0, 1]$  is continuously estimated as in Eq. (2.13) and scales the local quantization error relative to the diameter of the union of observed inputs.

$$d(t) = \min\left(\frac{E_q(t)}{S}, 1\right) \quad (2.13)$$

where  $S$  is computed as in Algorithm 1, which approximates the diameter of the input space learned until  $t$ . The function  $diam(\cdot)$  gives the diameter of a set, i.e., the largest distance between any two members of the set according to the Euclidean distance. Therefore, if  $d(t)$  is large, the network fits the data poorly and needs large readjustments. Conversely, if  $d(t)$  is small, the fit is likely to already be satisfactory for that input and no large updates are necessary at time  $t$ .

The neighborhood size is determined by  $d(t)$  as in Eq. (2.14); the neighborhood range  $\gamma$  is an upper bound for the neighborhood size, and does not change during training.

$$\Theta(d(t)) = \gamma \ln(1 + d(t)(e - 1)) \quad (2.14)$$

where  $\ln(\cdot)$  is the natural logarithm and  $e$  is the *Euler* number. Note that the scaling factor  $(e - 1)$  is chosen to ensure that the range of  $d(t) \in [0, 1]$  maps into the range of  $\Theta = [0, \gamma]$ .

**Algorithm 1:** PLSOM input space diameter estimation algorithm.

---

```

1 begin
  /* Let  $n = d + 1$ , where  $d$  is the dimensionality of the input
    space, since  $d + 1$  is the smallest number of vertices
    that can span a  $d$ -dimensional volume. */
2    $n \leftarrow d + 1$ 
3    $\mathcal{A} \leftarrow \emptyset$ 
4    $S \leftarrow -1$ 
5   foreach  $\mathbf{x}(t)$  do
6      $s \leftarrow \text{diam}(\mathcal{A} \cup \mathbf{x}(t))$ 
7     if  $s > S$  then
8       while  $\text{size}(\mathcal{A}) > n$  do
9         Remove the element of  $\mathcal{A}$  that is closest to  $\mathbf{x}(t)$ 
10      end
11       $\mathcal{A} \leftarrow \mathcal{A} \cup \mathbf{x}(t)$ 
12    end
13  end
14 end

```

---

The value of  $\Theta$  is used in the PLSOM neighborhood function:

$$h'_{ck} = e^{-\frac{1}{\Theta} \frac{\|r_c - r_k\|^2}{d(t)^2}} \quad (2.15)$$

Finally, the PLSOM update rule is given by Eq. (2.16).

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + d(t) h'_{ck}(t) [\mathbf{x}(t) - \mathbf{w}_k(t)] \quad (2.16)$$

If we compare the above update rule with the of the original SOM in Eq. (2.4), the PLSOM learning rate is given by the fit  $d(t)$  and the later is also used to scale the neighborhood radius. To compute the diameter of the input, it has to store some observations, which is some sense violate the data stream model (Section 2.3.1). Also, with evolving data the “diameter” of the input space may change. Although having only one parameter, it is problem-dependent, i.e., rarely one chosen value performs well across more than one data stream. In (Berglund 2010) no evaluation was performed regarding the estimation of this parameter besides trial and error.

## Dynamic SOM

The *Dynamic Self-Organizing Map* (DSOM) (Rougier and Boniface 2011) is also a variation of the SOM algorithm, where the original time-dependent (learning rate and neighborhood) learning function is replaced by a time-invariant one. It uses a constant learning rate and a constant *plasticity* parameter. The update rule of the DSOM is formalized in

Eq. (2.17):

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta \|\mathbf{x}(t) - \mathbf{w}_k\| h''_{ck}(t) [\mathbf{x}(t) - \mathbf{w}_k(t)] \quad (2.17)$$

$$h''_{ck} = e^{-\frac{1}{\epsilon^2} \frac{\|\mathbf{r}_c - \mathbf{r}_k\|^2}{E_q(t)^2}} \quad (2.18)$$

where  $\eta$  is a constant learning rate and  $\epsilon$  is the elasticity or plasticity parameter, which is also constant. If  $\mathbf{x}(t) = \mathbf{w}_c$ , then  $h''_{ck} = 0$ . Therefore, the magnitude of the updates (learning rate) is dependent on the distance between the prototype and the input pattern, while the range (neighborhood size) depends on the distance of the later to the BMU. This update scheme reflect two main ideas: (i) if a neuron is close enough to the data, there is no need for others to learn anything, the BMU can represent the data, and; (ii) if there is no neuron close enough to the data, any neuron learns the data according to its own distance to the data. The later idea is maybe somewhat inspired by the NG algorithm, where the magnitude of the updates is determined by the ordered rank of distances to the BMU. This works well for NG, but only because it does not impose neighboring relations between nodes. In DSOM they continue to exist and by adopting such strategy, a lot of topological defects can arise mainly during the unfolding of the map. The former idea seems logical, but is also very sensible to the initialization of the prototypes as discussed in the next paragraph. Finally, the parameter-space of DSOM is smaller than the original SOM, i.e., only 2 parameters, but from several tests that were performed they are also problem-dependent. No particular combination of values performs satisfactorily on more than one data stream.

From the above presentation of PLSOM and DSOM we can see they are similar in most aspects. Both implicitly estimate learning parameters based on the local error  $E_q(t)$ , but this makes these variants very sensible to the random initialization of the prototypes, e.g., if they initially cover the underlying distribution, but in an unordered manner, the magnitude of the updates will be small and the unfolding of the map hindered. The only way to overcome this is to use higher values in their parameters to force bigger initial updates. However, as they are constant, the lattice will never be able to stabilize over the distribution, i.e., prototypes fluctuate around their positions. Moreover, even with optimal parameters (very hard to determine) they fail to match the density of the input space.

### 2.5.3 Evaluation

Table 2.1 summarizes the fulfillment of the requirements established in Section 2.5.1 by the previous surveyed variants. Regarding the Original SOM algorithm, as described, the learning parameters are time-dependent, i.e., uses a time-dependent annealing procedure to monotonically reduce learning parameters and ensure convergence; possesses

Requirement	SOM Variant					
	Original SOM	GCS	ESOM	ESOINN	PLSOM	DSOM
Fixed topology	+	-	-	-	+	+
Time-Invariant parameters	-	+	+	-	+	+
Proper convergence	+	+	+	+	-	-
Incremental learning	+/- (but time dependent)	+	+	+	+	+
Compactness of model	+	-	-	-	+	+
Predicts evolving data	-	+	+/- (does not forget)	+	+/-	+/-
Robustness to noise	+/- (depends on learning rate)	+	+	+	-	-
High-dimensional data	+	+	+	+	+	+

Table 2.1: Evaluation of SOM variants under proposed requirements. Symbols (+), (-) and (+/-) mean the requirement (is), (is not) or (is partially) met, respectively.

a compact model in the sense that the size of the codebook is fixed and independent of the number of examples it processes; robustness to noise is not guaranteed when the learning parameters have high values.

Variants that do not keep the regular topology of the original SOM, i.e., GCS, ESOM and ESOINN, seem attractive from a pure VQ perspective, but they all assume limitations in visualizing the models and must resort to other algorithms (Deng and Kasabov 2003), e.g., Sammon's mapping (Sammon 1969).

Finally, fixed topology variants, e.g., PLSOM and DSOM, maintain the regular lattice structure, but do not converge properly in the sense of a VQ procedure. By failing to do so, visualizations may also be hindered. Also, using the localized error to estimate learning parameters is a simple approach, but does not provide a global measure of the model adaptation to the underlying data. It is important to note that in both proposals, authors also never address the visualizations of the obtained models.

Any distance-based clustering or VQ algorithm is affected by the curse of highdimensional data. However, most SOM models are deemed suited for high-dimensional data, given its VP capabilities (Penn 2005).

Given none of the above proposals satisfies the requirements established for self-organizing maps in dealing with the characteristics of data streams and allowing visual knowledge discovery, we can see that there is room for improvement in proposing new methodologies and/or variants that are better suited for the established requirements.

## 2.6 Current Methods for Cluster Analysis on Data Streams

The *data stream clustering* problem is defined as to maintain a continuously consistent good clustering of the sequence observed so far, using a small amount of memory and time (Gama 2010). The issues are imposed by the continuous arriving data points, and the need to analyze them in real time. These characteristics require incremental clustering and maintaining cluster structures that evolve over time, i.e., the data stream may continuously evolve, and new clusters might appear, others disappear, reflecting the dynamics of the stream.

Most of the work in incremental clustering of data streams has been concentrated on clustering observations rather than feature clustering. Clustering features (e.g., time series) is a very useful tool for some applications, such as sensor networks, electrical power demand, stock market, etc. The basic idea behind clustering streaming time series is to find groups of features that behave similarly through time.

### 2.6.1 Clustering Observations

Most current approaches seem to have settled on a two-phase approach to the problem: an *online* phase to abstract the incoming stream, i.e., maintaining summarizations, and; an *offline* phase performed at user request where traditional clustering methods (Section



2.4.7) are applied over the previous summarizations. Most abstraction methods use some variation of the concept of *cluster feature* (Zhang, Ramakrishnan, and Livny 1996).

According to the taxonomy presented in (Silva et al. 2013), existing methods can be characterized by the way they abstract the incoming stream (abstraction method), how do they deal with non-stationarity of data (window model) and the clustering algorithm that is used. The latter determines the shape of clusters that can be detected and if the goal is traditional example clustering or feature clustering. *CluStream* (Aggarwal et al. 2003) is a landmark publication in the sense that it popularized the two-phase approach to data streams, combining an online summarization of the incoming stream with an offline cluster analysis; it finds its roots in *BIRCH* (Zhang, Ramakrishnan, and Livny 1996). Regarding feature clustering methods in a stream setting the number of proposals is very scarce, e.g., (Rodrigues, Gama, and Pedroso 2008), and must employ different strategies from the ones found in “traditional” clustering.

Table 2.2 presents a set of proposed methods that perform cluster analysis over data streams, ordered historically. It summarizes the characteristics of the following methods, regarding the used taxonomy: *BIRCH* (Zhang, Ramakrishnan, and Livny 1996), *Scalable k-means* (Bradley, Fayyad, and Reina 1998), *Single-pass k-means* (Farnstrom, Lewis, and Elkan 2000), *Stream* (Guha et al. 2000), *Stream LSearch* (O’callaghan et al. 2002), *CluStream* (Aggarwal et al. 2003), *DenStream* (Cao et al. 2006), *D-Stream* (Chen and Tu 2007), *SWClustering* (Zhou et al. 2008), *ODAC* (Rodrigues, Gama, and Pedroso 2008), *SWEM* (Dang et al. 2009), *ClusTree* (Kranen et al. 2011) and *StreamKM++* (Ackermann et al. 2012). This set illustrates the variety of combinations of methodologies used until today and a brief overview is given next in respect to methodologies involved. More comprehensive surveys regarding clustering data streams methodologies can be found in (Amini, Wah, and Saboohi 2014; Silva et al. 2013).

### 2.6.1.1 Abstraction Method

In short, the data abstraction procedure aims at summarizing the data stream into a compact representation that captures the properties of the underlying distribution. The summary technique is directed towards the offline algorithm that is applied for the cluster analysis.

Four major types of data structures used for data abstraction are listed below. The data abstraction procedure exploits the fact that the input space of the data stream is usually not uniformly occupied. Thereby, the problem of cluster analysis is reduced to the set of summaries, which is much smaller than the original data stream input space. Hence, the data stream is somewhat pre-clustered by summarizing the data stream into dense regions, according to the used data structure. It should be noted at this point that evolving data is dealt within the abstraction method, usually by means of a window model, discussed next.

Algorithm	Abstraction Method	Window Model	Cluster Algorithm	Cluster Shape	Cluster Problem
BIRCH	cluster feature	landmark	hierarchical	hyper-sphere	observation
Scalable k-means	cluster feature	landmark	k-means	hyper-sphere	observation
Single-pass k-means	cluster feature	landmark	k-means	hyper-sphere	observation
Stream	prototype array	landmark	k-medoids	hyper-sphere	observation
Stream LSearch	prototype array	landmark	k-medoids	hyper-sphere	observation
CluStream	cluster feature	landmark	k-means	hyper-sphere	observation
DenStream	cluster feature	damped	DBSCAN	arbitrary	observation
D-Stream	grid	damped	DBSCAN	arbitrary	observation
SWClustering	cluster feature	sliding	k-means	hyper-sphere	observation
ODAC	correlation matrix	landmark	hierarchical	arbitrary	feature
SWEM	cluster feature	damped	EM	hyper-ellipsis	observation
ClusTree	cluster feature	damped	k-means/DBSCAN	arbitrary	observation
StreamKM++	coreset tree	landmark	k-means	hyper-sphere	observation

Table 2.2: Overview of state-of-the-art algorithms for cluster analysis over data streams — adapted from (Silva et al. 2013).

**Prototype array** is a simplified summarization structure that, as the name implies, consists in an array of prototypes (e.g., centroids or medoids) that summarizes the data partition (Guha et al. 2000).

**Cluster features (CF)** is a compact representation of a set of points (Zhang, Ramakrishnan, and Livny 1996). A CF structure is a triple  $\{m, LS, SS\}$ , used to store the sufficient statistics of a set of points:  $m$  is the number of data points,  $LS$  is a vector of the same dimension of data points that store the linear sum of the  $m$  points and  $SS$  is a vector of the same dimension of data points that store the square sum of the  $N$  points. Let  $CF_1 = \{m_1, LS_1, SS_1\}$  and  $CF_2 = \{m_2, LS_2, SS_2\}$ . Cluster features are *incremental* and *additive*. If an observation  $x$  is added to the cluster, the sufficient statistics are computed as:

$$\begin{aligned} LS_1 &= LS_1 + x \\ SS_1 &= SS_1 + x^2 \\ m_1 &= m_1 + 1 \end{aligned}$$

and the additive property allows merging two clusters by the sum of their parts, as:

$$CF_1 + CF_2 = \{m_1 + m_2, LS_1 + LS_2, SS_1 + SS_2\}.$$

A CF-triple has sufficient information to define variances along the  $i$ th-dimension as  $(SS_i/m) - (LS_i/m)^2$  and the centroid as  $(LS/m)$ . Distances can also be easily computed from the available information.

One can think of a CF as a set of objects, but only the CF triple stored as summary. This CF summary is not only efficient because it stores much less than all the data objects in the CF, but also accurate because it is sufficient for calculating pertinent measurements needed to make clustering decisions. The use of clustering features is of major importance in current state-of-the-art algorithms for large data sets and/or streaming data, and some variations are found in literature. BIRCH maintains a height-balanced CF-tree as a summary structure. It is a very compact representation of the data stream because each entry in a leaf node is not a single data object but a sub-cluster. In *CluStream* this concept was extended to include temporal information (TCF) and called *micro-cluster* (Aggarwal et al. 2003). The TCF was combined with histogram information in *SWClustering* into a structure called Exponential Histogram of Cluster Features (EHCF).

**Grids** partition the  $d$ -dimensional space into density grid cells and each example is incrementally mapped onto a cell (Cao et al. 2006). This differs from previous structures, in the sense that cluster analysis algorithms that perform over grids are concerned not with the data objects but with the value space that surrounds the data objects. Hence, density-based algorithms are usually applied over grids.

**Coreset tree** is a binary tree where each node stores a mixture of sufficient statistics and prototypes, i.e., *coresets*. A *coreset* is a small weighted set of objects that approximates the original object set (Agarwal, Har-Peled, and Varadarajan 2005). The aim is to reduce  $2m$  objects (data objects or previous centroids) to  $m$  objects, by performing merge-reduce operations. This tree approximates the data points from the data stream regarding the k-means optimization problem.

### 2.6.1.2 Window Model

In a data stream scenario, recent information can reflect changes in the data distribution. This information can be used to explain the evolution of the process under observation. The established way of dealing with this non-stationarity of data is through *window models* (Gama 2010), which can be divided into three types:

**Sliding** window models imply that only the most recent information is stored in the data abstraction structures, whose size can be fixed or variable (Zhou et al. 2008). It can be seen as a *queue* and methods that use this model only update the summaries of the data inserted into the window, giving equal importance to each example.

**Damped** window models, or time-fading models, are similar to the sliding window model, but associate higher weight to newer data using an exponential decay function (Cao et al. 2006).

**Landmark** window models proceed by dividing the stream into disjoint portions (chunks), which are separated by landmarks (Farnstrom, Lewis, and Elkan 2000). Landmarks can be defined either in terms of time (e.g., daily or weekly basis) or in terms of the number of examples processed since the last landmark.

### Evolving Data

Overall, evolving data is dealt by maintaining the abstraction structures updated over a specific window model. In general, observations are absorbed by existing CFs/micro-clusters if they rely within a specified distance threshold. Otherwise, new ones are created. *BIRCH* limits the size of the CF-tree by available memory and resorts to merging “crowded subclusters into larger ones” — (Zhang, Ramakrishnan, and Livny 1996). It does not address “old information” because the CF does not possess temporal information, i.e., it was designed for incremental learning by generating an hierarchical cluster structure. Micro-clusters, on the other hand contain temporal information, and allow the deletion of old micro-clusters because “a given micro-cluster might correspond to a point of considerable cluster presence in the past history of the stream, but may no longer be an active cluster in the recent stream activity” — (Aggarwal et al. 2003). *CluStream* generates a pyramidal time frame of micro-clusters. In this technique, micro-clusters are stored at differing levels of

granularity depending upon the recency, until a specific maximum past horizon. This provides a trade-off between the memory constraints and the ability to recall summary statistics from different time horizons. Although this allows a “peek” into the past, it sacrifices the available expressiveness of the micro-clusters to abstract the current distribution. Differently, whenever a significant change happens in the stream’s distribution, *SWEM* re-distributes the set of micro-clusters in the entire data space by using split and merge operations. *D-Stream*, which uses a grid abstraction, also takes into account evolving data by assigning a decay counter to each grid cell.

None of the reviewed methods provide ways to explicitly detect changes in the distribution, i.e., signal some kind of warning to the user.

### 2.6.1.3 Cluster Algorithm

Some of the enumerated methods maintain a clustering result within the abstraction structure as approximations of the k-means optimization problem, e.g., *Scalable k-means*, *Single-pass k-means* and *StreamKM++*, or k-medoids, e.g., *Stream* and *Stream LSearch*, through landmark windows.

All other methods rely on summary structures to where traditional clustering algorithms are applied, offline. Comparison between the SOM and other traditional clustering methods was already discussed in Section 2.4.7, together with their strengths and weaknesses. As most overviewed methods apply traditional methods to summarization structures (see Table 2.2), the same strengths and weaknesses apply. *CluStream* proposed applying the k-means algorithm to the pyramidal time frame, containing the *micro-clusters*, allowing extracting of micro-clusters of different time horizons. *ClusTree* was tested with k-means and DBSCAN. Other surveyed methods that rely on DBSCAN also apply it to CF-like structures, e.g., *DenStream*, or to grid structures, e.g., *D-Stream*. BIRCH is an exception, as it maintains an hierarchical clustering of the CFs.

### 2.6.2 Clustering Features

The above methods are devised to perform traditional clustering of observations. On the other hand, one may be interested in clustering the features in a data stream setting. If we consider each feature of the data stream as an individual time-series, then the goal of an incremental clustering system for multiple time series is to find (and make available at any time  $t$ ) a partition  $P$  of those streams, where streams in the same cluster tend to be more alike than streams in different clusters. Let  $\mathbf{x} = \{x_1, x_2, \dots, x_d\}$  be the complete set of  $d$  data streams and  $\mathbf{x}(t) = \{x_1^t, x_2^t, \dots, x_d^t\}$  be the example containing the observations of all streams  $x_i$  at the specific time  $t$ . An example partition could be defined as  $P_t = \{\{x_1, \{x_3, x_5\}\}, \{x_2, x_4\}\}$ , stating that data streams  $x_1, x_3, x_5$  have some similarity between them (more pronounced between  $x_3$  and  $x_5$ ), being at the same time somehow more dissimilar from  $x_2$  and  $x_4$ .

One of the first works for this task was presented in (Rodrigues, Gama, and Pedroso 2008), with the *Online Divisive-Agglomerative Clustering* (ODAC) proposal. The ODAC is a feature clustering algorithm that constructs a hierarchical tree-shaped structure of clusters using a top-down strategy. The system uses the Pearson's correlation coefficient between time series as a similarity measure, but performs these computations over *sufficient statistics* of each time series, maintained over time — a variation of CFs.

### 2.6.3 Collaborative and Distributed Learning

Current ubiquitous environments require that data mining systems should be designed not to operate as a monolithic centralized application, but as a distributed collaborative process. Instead of centralized relevant data in a single server and perform the data mining operations later, the entire process should be distributed and paralleled throughout the entire network of processing units (Rodrigues and Gama 2014). A brief overview of proposed methods is provided, but more in-depth surveys can be found in (Gaber et al. 2014; Rodrigues and Gama 2014).

Consider a network of sensors. A traditional approach would consist of a centralized process that would gather data from sensors and then perform a clustering procedure. However, this raises two problems: first, the amount of data that each node needs to communicate is restrained by the available bandwidth; second, and consequently, as the number of sensors grow, the scalability of the approach degrades. In a different two-level approach, as sensors evolve to “smart” sensors with processing abilities, each sensor can separately cluster its own data and then combine the results on a centralized process which defines the final clusters, based on the models transmitted by each sensor. This has the advantages of clustering models becoming assets of the respective sensors and limits data to be transmitted; this is similar to a strategy of cluster ensembles (Strehl and Ghosh 2003).

In (Datta et al. 2006), a distributed k-means clustering technique is presented: local algorithms monitor the data distribution and when they signal change, data is centralized and centroids updated; the new centroids are pushed back to the local nodes. A different approach, but with local and global computations is proposed in (Cormode, Muthukrishnan, and Zhuang 2007; Gama, Rodrigues, and Lopes 2011), with the same overall goal. Also, in (Kargupta et al. 2001) the same strategy was previously devised with PCA. The disadvantage of these approaches is that they all depend on a centralized clustering procedure.

On the other hand, more interesting local and global clustering approaches have been proposed, namely (Klusck, Lodi, and Moro 2003) for density-based clustering and (Bandyopadhyay et al. 2006) for k-means. These techniques involve producing local models at each node, that are later combined with other nodes (through P2P networks) or centralized — this is considered the best strategy to be pursued in this thesis.

Although collaboration is implicit in the previous proposals, it can be better understood in a mobile setting. The interest is on how the knowledge available in the community can be integrated in local models to improve them; the goal is not to learn a global model, but to learn from other devices their models, while maintaining a local or subjective perspective. For example, (Wurst and Morik 2007) explore this idea by investigating how communication among peers can enhance individual local models without aiming at a common global model, in distributed media organization platform. The motivation is similar to what is proposed in transfer learning (Pan and Yang 2010), assuming, however, a batch scenario.

Nonetheless, leveraging the trade-off between local and global models in ubiquitous environments, either through centralization or collaboration, is dependent of the local processing of data streams, hence the emphasis was given to this primary problem.

## 2.7 Critical Overview

The fundamental purpose of this chapter was to provide answers to the initial research questions (Section 1.1), namely:

**RQ1** *What are the limitations of Self-Organizing Maps regarding streaming data?*

**RQ2** *Can Self-Organizing Maps provide a valuable tool for data stream cluster analysis regarding current methods? If so, which research paths should be pursued in terms of relevant contributions?*

**RQ3** *How to address the use of Self-Organizing Maps in ubiquitous environments?*

In respect to **RQ1**, an overview was given regarding the original SOM algorithm and its limitations in processing data streams, in particular the time-dependent learning parameters that affect the incremental learning capability of the algorithm in response to non-stationary environments. Although the original *Online* SOM algorithm allows incremental processing of observations, it was devised for *static* data, where  $N$  is known in advance, and the PDF of the underlying distribution is considered stationary. In Section 2.5 requirements for SOM variants dealing with the characteristics of data streams were established. Some derive from constraints imposed by the data stream model and, consequently, general characteristics algorithms should provide in performing cluster analysis from data streams (Section 2.3.1). The remaining requirements are SOM-related, i.e., maintaining the strengths of the original SOM in performing exploratory knowledge discovery (Section 2.4.5), and to remove the time dependency when estimating learning parameters. Most surveyed proposals (Section 2.5.2) deal with non-stationarity by adding or removing neurons from the network, e.g., *GCS*, *ESOM*, *ESOINN* — this alters the regular topology of the SOM lattice and makes the application of the visualizations



impossible. Others, e.g., *PLSOM* and *DSOM*, maintain the fixed topology and estimate learning parameters through the instant quantization error  $E_q(t)$ . While this allows the SOM to retain an indefinite plasticity, this strategy does not allow proper convergence of the prototypes towards a density matching vector quantization of the input space, which also hinders the effectiveness of the visualizations. Hence, there is room for improvement concerning new variants of the SOM algorithm specifically tailored for cluster analysis over data streams. As an initial hint, such algorithms should include some kind of global assessment metrics to gauge the trend of the learning process towards the current underlying distribution of the data stream, e.g., global error estimates. With this, learning parameters should be kept high if this error trend is increasing (the distribution is drifting from the current mapping of the prototypes) and allow the map to globally adjust to the new distribution. On the other hand, if the error trend is decreasing (the model is converging) the learning parameters should decrease monotonically to allow proper convergence of the map.

In respect to **RQ2**, most current proposals have, inspired by *BIRCH* and popularized by *CluStream*, settled in a two-phase approach to data streams: (i) *online* summarization of the data stream, and; (ii) *offline* cluster analysis. Most attention is focused on efficient abstraction structures and methods that summarize the data stream. Mechanisms to deal with change are embedded into these methodologies, following particular window models. In the context of data streams and limited memory it is impossible to summarize an infinite time horizon. Therefore, abstraction procedures also aim at forgetting past summaries that represent past persistent states (stationary moments) of the underlying distribution. If, somehow, change detection mechanisms can be introduced, this may allow to store past summary models for later evolutionary cluster analysis (Chakrabarti, Kumar, and Tomkins 2006), if desired. However, none of the surveyed proposals address this scenario. This approach contrasts with the *CluStream* approach to keep summaries over a longer timespan. In the offline stage, current proposals provide clustering results, at user request, through modifications of hierarchical, partitioning (e.g., k-means) or density-based (e.g., DBSCAN) algorithms, being the later two the most common, applied to the summary structures.

In Section 2.4.7 a comparative analysis between the SOM and other traditional clustering methods was presented regarding clustering results and knowledge discovery. Undoubtedly, the SOM is far more attractive for purposes of visual and exploratory knowledge discovery, one of the main reasons for its popularity. The SOM is widely used as a tool for mapping high-dimensional data into a two-dimensional representation space. This mapping retains the relationship between input data as faithfully as possible, thus describing a topology-preserving representation of input similarities in terms of distances in the output space. The main advantage of such a mapping is the ease by which a user gains an idea regarding the structure of the data by analyzing the map. Through visualization techniques readily obtained from maps, it is possible to visually identify clusters



(and cluster descriptions) on the map, without imposing any notion and number of clusters. This characteristic is not found in other current data stream clustering methods. K-means imposes previous knowledge of the number of clusters, which can vary throughout the data stream; it also assumes the clusters are spherical and of equal importance. Density-based algorithms depend on the parameterization of the required inter-distance of a set of data points to consider them a cluster; but with high-dimensional data all points are somewhat far from each other. However, in a streaming environment what is the relevance of the information we can obtain from these methods, where timely knowledge discovery is even more critical, apart from a grouping of summarized observations? In streaming environments maximum knowledge should be obtained in the least amount of time, e.g., monitoring applications. Here is where we should realize the SOM becomes interesting. Also, apart from parameterization, the SOM exploratory knowledge discovery is readily interpretable by the everyday person, if we intend to make it available to anyone besides experts.

Cluster analysis over data streams begs for fast and timely knowledge discovery. Apart from SOM and its visualization abilities, other methods involve some kind of trial and error approach to obtain pertinent results. In ([“Visual Data Mining” 2005](#)) strong and compelling arguments are made regarding visual knowledge discovery, quoting: *“Visualization has proven to be an essential support throughout the KDD process in order to extract hidden information from huge amount of data. Visualization techniques enable the direct integration of the user to overcome major problems of automatic machine learning methods such as presentation and interpretation of results, lack of acceptance of the discovered findings or limited confidence in these. Since computers are still much less useful than the ability of the human eye for pattern matching, visual data exploration techniques provide the user with graphic views or metaphors that represent potential patterns and data relationships. (...) Human perception can identify data relationships when a data set is two or three-dimensional. However, multidimensional data sets with more than three dimensions require some kind of visual transformation to be explored and analyzed.”* We cannot avoid considering the SOM as a realization of these arguments.

From the above discussion, motivation for the use of SOMs in a streaming setting should be clear. Moreover, current methods do not possess the versatility to cluster both observations and features. Most of the literature is devoted to clustering of streaming observations, whereas literature regarding clustering the features, e.g., time-series, is scarce. The SOM also has the ability to detect correlated features through component planes. In respect to cluster analysis of data streams, an overall depiction of the current state-of-the-art and the identified research gap is depicted in [Figure 2.10](#).

Finally, regarding **RQ3**, the trade-off between *local* and *global knowledge* is now the key point for clustering procedures over *ubiquitous* data streams ([Rodrigues and Gama 2014](#)). If data streams, pertaining the same problem, are being generated at distributed

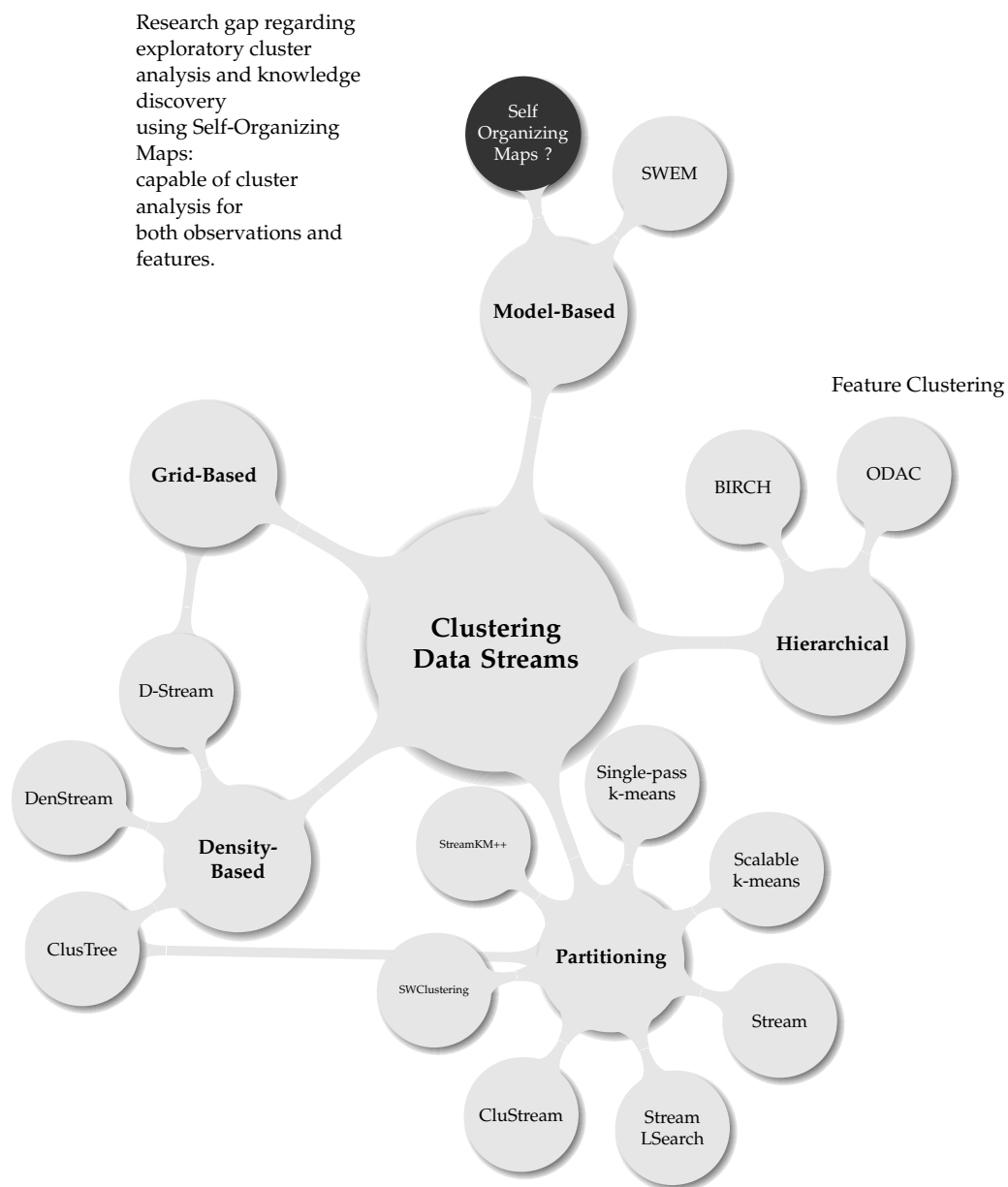


Figure 2.10: Panorama of available methods for clustering data streams before research.

sites or devices, the favored approach is to allow the existence of local modes that capture the subjective view (clustering result) of the problem. Then, these models can be centralized and/or shared to generate (more) global models in order to obtain a global view of the problem. Having a SOM model that can produce a local clustering result, the prototypes themselves describe the underlying distribution in a very compact way and can be transmitted to other sites or devices to produce (more) global SOM models.

### 2.7.1 Research Paths and Additional Research Questions

Directed towards the above research questions, the following research paths were identified. In respect to using the SOM over data streams, two approaches can be taken, namely:

**Two-Phase Approach.** First, inspired by the popular two-phase approach to data streams, generate SOM models from summaries of the data stream. SOM models can potentially be generated from existing abstraction procedures using CF-like structures or grid summaries. However, these are generally very reduced in their number and SOM needs a bigger pool of observations. If we intend to keep methodology proposals in the ANN domain, ART networks seem attractive for this purpose (please recall Figure 2.7c for an example of generated ART categories that are able to summarize a distribution). The ART2-A algorithm (Section 2.4.1.2) predicts evolving data (proposed to solve the stability–plasticity dilemma) in some extent by creating new categories when the existing are too different — based on a distance threshold, for the current observation. But two problems remain: first, the computation of this threshold is, however, dependent of the vigilance parameter, that when poorly estimated can lead to the proliferation of categories (high vigilance), violating the compact model requirement, or very coarse data representation (low vigilance) leading to a poor abstraction of the input space; second, a means to allow forgetting of old information must also be addressed. Noise will be treated by ART as new categories, hence not being robust to noise, without further modifications. Nonetheless, with a sufficient number of categories (composing a static dataset) it should be possible to use the original SOM algorithms in an offline stage and perform exploratory knowledge discovery.

Therefore, this research path raises the following research question:

- *Can ART networks produce an efficient data stream summary from where SOM models can be generated?*

**Online Approach.** This approach is more ambitious and addresses a SOM variant tailored for non-stationary data streams, obeying the requirements established in Section 2.5. Due to the topological ordering of the SOM prototypes and high-dimensional data projection abilities, the SOM may be the most compact way to abstract a data distribution, with immediate visualization capabilities. Global assessment metrics should be devised

from where learning parameters are monotonically increased or decreased dependent on the adaptation of the SOM lattice to the underlying distribution. This proposal has the benefits of allowing real-time knowledge discovery from data streams.

Consequently, the additional research question this research path raises is the following:

- *Can Self-Organizing Maps learn non-stationary data streams, while keeping the original SOM properties intact?*

**Distributed/Collaborative Learning.** Having achieved the later approach, the collaborative/distributed aspect can be more easily tackled. Considering summary models represent the local “knowledge” acquired in a particular site/device, obtaining global models imply the “merge” of these local models. However, we should not transmit all prototypes of the codebooks, since some of them may be “unrepresentative” prototypes, that do not actually represent the local underlying distribution (recall Section 2.4.5, illustrating the non-optimal VQ procedure of the SOM). In a distributed collaborative setting, we can envision mobile devices with sensing capabilities sharing their models with other passerby devices, in an attempt to obtain global knowledge in a shared physical environment. In a centralized approach, we can imagine a large network of monitoring sensors deployed in a city, e.g., monitoring air quality, where local models describe a particular location and a global model depicts the entire city.

The additional research question raised in this setting is the following:

- *Can distributed and collaborative learning strategies over ubiquitous data streams be devised using Self-Organizing Maps?*

**Real-World Applications.** Finally, from the fulfillment of the above research directions, a set of real-world problems that could benefit from the research should be presented. Hence, the final research question raised is:

- *What sort of real-world problems can benefit from this research?*

These additional research questions guide the main contributions made in this thesis and are the focus of the following chapters.



## Aims and Methodology Overview

The scientist is not a person who gives the right answers,  
he's one who asks the right questions.

---

CLAUDE LÉVI-STRAUSS, FRENCH ANTHROPOLOGIST  
(1908-2009)

This thesis addresses the use of Self-Organizing Maps in a data stream setting, also contemplating their ubiquitous or distributed aspect. Such approach was motivated in the previous chapter, where the use of Self-Organizing Maps was deemed relevant as a versatile model-based exploratory cluster analysis method. From obtained SOM models, and in particular the U-Matrix visualization, traditional cluster analysis (clustering observations) can be performed, without assumptions regarding the shape and number of clusters. Also, a description the detected clusters can be obtained through the component planes visualizations. As seen, these later visualizations can be used to detect correlated features, consequently allowing the SOM to additionally perform clustering of features, which, in a stream setting, is closely related to time-series clustering.

The following four sections concern the additional formalized research questions established in Section 2.7.1, their aims and methodology summary — each of these sections introduce the following four chapters, respectively. After, some considerations regarding the normalization of data streams and assumptions made throughout the proposals of this thesis are discussed in Section 3.5. Finally, artificial data streams used in evaluating various algorithms and methodologies are briefly described in Section 3.6.

### 3.1 A Two-Phase ART/SOM Approach to Data Streams

As surveyed in Section 2.6.1, most current state-of-the-art approaches to clustering data streams have settled in a two-phase approach: an *online* summarization procedure coupled with an *offline* cluster analysis stage at user request. The established goal is to investigate the feasibility of such approach based on an *online* data stream abstraction by ART networks, towards *offline* SOM model generation. As an additional secondary goal, to devise an assessment metric of fit regarding the current abstraction and the underlying distribution of the data stream.

#### Research Question

Can ART networks produce an efficient data stream summary from where SOM models can be generated?

#### Aim

Perform cluster analysis over non-stationary data streams.

#### Objectives

- Achieve summarizations of data streams through an ART algorithm;
- Produce SOM models on-demand from the previous summarizations;
- Test whether the mean quantization error metric can be adapted to an evolving codebook for model assessment.

#### Methodology Overview

Establishment of the *micro-category* as the summarization basic structure. Micro-categories are continuously generated over a landmark window through a constrained ART2-A algorithm to produce an evolving codebook. The Batch SOM update rule is adapted to take into account information from micro-categories. A set of micro-categories is composed, regarding a specified elapsed time interval of the data stream. The *average quantization error* is established as a primary metric to assess model adaptation by averaging local quantization errors in a sliding window. Parameter sensitivity analysis is performed for the introduced parameters of the summarization procedure. Evaluation of proposed methodology is carried out using artificial data streams.

## 3.2 The Ubiquitous Self-Organizing Map for Non-Stationary Data Streams

The SOM is probably one of the best summarization structures available, from where knowledge discovery can be readily derived through its visualizations. However, current SOM variants, that would be fit for data streams, lack the combined desired properties proposed in Section 2.5.1. Of particular interest is to increase or decrease the learning parameters proportionally to the trend of the model adaptation to the underlying distribution of the data stream. The established goal is to investigate if such an algorithm can be realized by estimating learning parameters from global assessment metrics of fit between the current codebook and the underlying non-stationary data stream. Devising an aforesaid variant allows real-time exploratory cluster analysis (and knowledge discovery, in general) by means of the SOM visualizations. A second goal regards proposing an automatic feature clustering method for SOM models.

### Research Question

Can Self-Organizing Maps learn non-stationary data streams, while keeping the original SOM properties intact?

### Aim

Develop a new variant of the SOM tailored for non-stationary data streams and real-time exploratory cluster analysis.

### Objectives

- Derive global assessment metrics of fit for the SOM regarding the underlying data stream;
- Modify the SOM update rule to estimate time-independent learning parameters from the above metrics;
- Introduce a feature clustering method based on component planes;
- Evaluate the capabilities of the obtained algorithm for performing exploratory cluster analysis over observations and features.

### Methodology Overview

The *average quantization error* and *average neuron utility* metrics are proposed and combined in a *drift function* that is used to estimate learning parameters. After an initial ordering phase over the data stream, the remaining of the learning procedure is guided solely based on the previous function — the algorithm implements a finite state machine composed by two states corresponding to the *ordering* and *convergence* phases suggested

by Kohonen (please recall Section 2.4.4); the algorithm transitions between both states depending on specific conditions. The algorithm is named *Ubiquitous Self-Organizing Map* (UbiSOM), already predicting applications for ubiquitous data streams. Parameter sensitivity analysis for two newly introduced parameters is performed. Comparison of the UbiSOM against the Original SOM algorithm, PLSOM and DSOM is presented. Finally, evaluation of the algorithm (and methodology) in performing cluster analysis over observations and features is presented. All results are obtained using artificial data streams.

### 3.3 Distributed and Collaborative Learning of Ubiquitous Data Streams

Given the possible distributed nature of data streams in some applications, the established goal concerns addressing the trade-off between local and global models, leveraging the capabilities of the UbiSOM in maintaining models over local data streams.

#### Research Question

Can distributed and collaborative learning strategies over ubiquitous data streams be devised using Self-Organizing Maps?

#### Aim

Distributed and collaborative learning of ubiquitous data streams.

#### Objectives

- Introduce a mechanism to filter out unrepresentative prototypes from UbiSOM codebooks;
- Devise methodology to transmit and centralize UbiSOM codebooks and allow the generation of global models;
- Differentiate the above strategy from collaborative learning, where devices interacting over the physical space are able to share models.

#### Methodology Overview

Both distributed and collaborative learning methodologies involve the communication or exchange of “filtered” UbiSOM codebooks, from where unrepresentative prototypes are removed. These codebooks are then centralized or shared between devices to compose dynamic datasets, from where the global models are generated; a simple method to remove similar prototypes is also introduced as a way to avoid overvaluing shared regions of the input space. Evaluation of proposed methodologies is performed using artificial data streams.



### 3.4 Real-World Applications

Algorithms and methodologies are only as useful as their real-world applicability. The established goal is to investigate the potential of the proposed algorithms and methodologies in a set of real-world problems.

#### Research Question

What sort of real-world problems can benefit from this research?

#### Aim

Real-world applications of proposed algorithms and methodologies.

#### Objectives

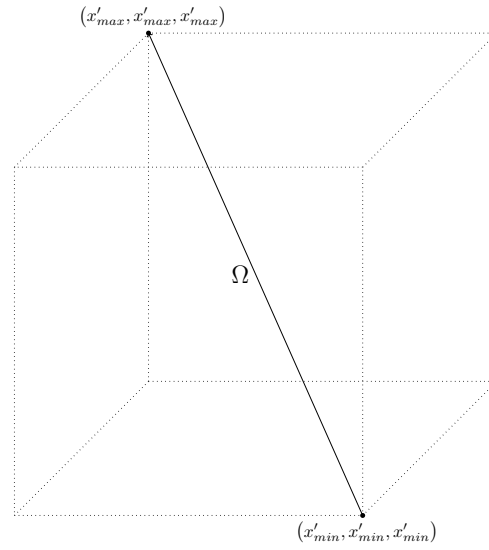
- Selection of a real-world problem from where to perform traditional cluster analysis over data streams;
- Use real-world distributed data streams to illustrate the generation of global models;
- Perform feature clustering over a real-world data stream.

#### Methodology Overview

Application of the UbiSOM algorithm to an household electric consumption data stream to continuously provide a clustering result of electric consumption patterns. A similar application to distributed air quality monitoring data is presented, but with local UbiSOM models generated at each monitoring station and global models generated to assess air quality in the overall area. Finally, an application to stock market data, to illustrate the potential of the feature clustering abilities of the SOM, is presented using the StreamART2A/SOM methodology. The goal is to cluster similarly behaved time series, that can be later used for portfolio selection and/or monitoring.

### 3.5 Considerations on the Normalization of Data Streams

Any distance-based clustering algorithm is sensible to the numerical range of features. It is a well-known fact that performing normalization has a huge impact on the quality of a clustering (or VQ) process. This is due to the distance computations, where some features can dominate the results, e.g., when they have a higher dynamic range. Normalization is then suggested to avoid this problem (Shalabi, Shaaban, and Kasasbeh 2006). Regarding data streams, no clear discussion of this problem was found in literature; in general, works assume the normalization. One should note that there are “range-insensitive”



Maximum distance in the hypercube is  $|\Omega|$

Figure 3.1: Maximum distance in the assumed normalized input space of data streams.

metrics, e.g., the dot product, which can be used in clustering tasks, but are not useful for the purposes of this research, in the sense that they do not permit the generation of prototypes of data (abstraction of the input space).

Therefore, proposed methodologies in the following chapters assume that a *min-max* normalization can be performed on the data streams. This type of normalization scales all features within the same range, based on their minimum and maximum values, and usually yields the best results in most data mining tasks (Shalabi, Shaaban, and Kasasbeh 2006); it is also easy to perform over incoming observations in an online fashion. To clarify, the assumptions made regard knowing beforehand lower and upper bounds for the features, allowing the normalization to be performed. With this information, we can derive the maximum distance possible between two examples in the normalized input data space, denoted by  $|\Omega|$ . It is common for this normalization to scale features in the range  $[0, 1]$ , but this is not strictly necessary. In general, if all features are normalized and bounded between  $[x'_{min}, x'_{max}]$ , this yields a hypercube, where the maximum distance between any two points is given by Eq. (3.1) and illustrated in Figure 3.1. This can be seen as the largest diagonal of the normalized input space.

$$|\Omega| = (x'_{max} - x'_{min}) \sqrt{d}. \quad (3.1)$$

The value  $|\Omega|$  is later used to maintain Euclidean distances in the interval  $[0, 1]$  and applied in particular situations, e.g., easily translate between ART vigilance  $\rho \in [0, 1]$  and distance thresholds, and maintain global assessment metrics of the UbiSOM also in the  $[0, 1]$  interval.

In real-world applications, computing  $|\Omega|$  requires that lower and upper bounds of individual features values are known. In sensor data, for example, data is bounded to the range of the sensor outputs themselves. In financial data, one can establish bounds from historical information or within a window model (Ogasawara et al. 2010). Further research on this subject is traversal for data streams.

### 3.6 Artificial Data Streams

In the following three chapters several artificial data streams are used to evaluate algorithms and methodologies. Table 3.1 summarizes these data streams together with reference to chapters where they are used. Below a brief description of each and intended usage is given — note that, besides a specific aim, all data streams (except *Correlated*) were used in the parameter sensitivity analysis of algorithms in Chapters 4 and 5.

Except data streams *d2k20*, *d3k20* and *d5k20* — obtained from publicly disclosed data in (Frahling and Sohler 2008), all others were generated for this research study. Repository information and illustrative images of these data streams can be found in Appendix B.

**Gauss** Two-dimensional stationary data stream containing 100 000 points from a bi-variate normal (Gaussian) distribution, centered at the origin and identity co-variance matrix; used to evaluate if proposed and compared methods can model the input space density properly;

**Complex** Two-dimensional stationary data stream containing 100 000 points describing a complex cluster structure, with seven clear clusters; used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models. It is also split into four quadrants when illustrating the distributed and collaborative learning methodologies;

**Chain** Three-dimensional stationary data stream containing 100 000 points describing two interlocked rings (clusters); used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models;

**d2k20, d3k20 and d5k20** Stationary data streams consisting of 300 000 points in  $d$  dimensions, describing 20 Gaussian clusters in different dimensions. The data stream is used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models; it is also used to test scalability of algorithm in Chapter 4.

**AbruptOneTwo** Two-dimensional non-stationary data stream containing 200 000 points describing an abrupt-changing (change point at  $t = 50\,001$ ) simple cluster structure with one Gaussian cluster splitting in two; aims at evaluating how different algorithms react to sudden change in the distribution;

Name	$d$	$N$	Stationary	Change at	#Clusters	Chapters
<i>Gauss</i>	2	100 000	Y	-	1	4, 5, C
<i>Complex</i>	2	100 000	Y	-	7	4, 5, 6
<i>Chain</i>	3	100 000	Y	-	2	4, 5, C
<i>d2k20</i>	2	300 000	Y	-	20	4, 5, C
<i>d3k20</i>	3	300 000	Y	-	20	4, 5, C
<i>d5k20</i>	5	300 000	Y	-	20	4, 5, C
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2	4, 5, C
<i>DriftTwoOne</i>	2	200 000	N	[50 001, 150 000]	2/1	4, 5, C
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2	4, 5
<i>Hepta</i>	3	150 000	N	100 001	7/6	4, 5
<i>Correlated</i>	10	100 000	N	50 001	NA	5

Table 3.1: Summary of artificial data streams, where  $d$  is the dimensionality of the data stream and  $N$  is the number of observations.

**DriftTwoOne** Two-dimensional non-stationary data stream containing 200 000 points describing a gradual change in a simple cluster structure, i.e., two *Gaussian* clusters merging into one (change occurs during  $t = [50\,001, 150\,000]$ ); aims at evaluating how different algorithms react to gradual change in the underlying distribution;

**Clouds** Two-dimensional non-stationary data stream containing 200 000 points describing a gradual changing cluster structure of three *Gaussian* clusters (change occurs during  $t = [50\,001, 150\,000]$ ); aims at evaluating how different algorithms react to gradual change in the underlying distribution;

**Hepta** Three-dimensional non-stationary data stream containing 150 000 points describing an abrupt-changing cluster structure (change point at  $t = 100\,001$ ). From the initial seven *Gaussian* clusters (hence the name), one disappears; aims at evaluating how different algorithms react to sudden change in the underlying distribution and to motivate the proposal of the *average neuron utility* assessment metric in Chapter 5;

**Correlated** As opposed to the previous data streams, this is utilized to evaluate the feature clustering methodology, i.e., time-series clustering. The data stream contains 5 pairs of correlated time-series ( $d = 10$ ) with different degrees of correlation. The pairwise correlations abruptly change at  $t = 50\,001$  and are described in Chapter 5 and also in Appendix B.

# 4

## A Two-Phase ART/SOM Approach to Data Streams

Simplicity is the ultimate sophistication.

---

LEONARDO DA VINCI, ITALIAN POLYMATH (1452-1519)

The SOM is a valuable tool in knowledge discovery with numerous applications found throughout literature. However, its adaptation to a streaming context is far from straightforward. On the other hand, ART networks exhibit a natural incremental learning capability that can be explored to provide an *online* abstraction of the data stream, with the intent to generate *offline* SOM models for exploratory cluster analysis. The goal is to replicate the popular two-phase approach to data streams within the ANN field.

### 4.1 Chapter Overview

This chapter addresses a two-phase ART/SOM approach to data streams through the proposal of the *StreamART2A/SOM* methodology. The *StreamART2A* is a data stream abstraction procedure that operates *online*; from this abstraction, *offline* SOM models are generated. The (moving) *average quantization error* metric is also introduced in this chapter as a primary metric of fitness between a competitive learning codebook and the underlying distribution.

The chapter is organized as follows. Section 4.2 presents the motivation and aim for the overall proposal. The two-phase approach is described in Section 4.3, with particular attention devoted to the *StreamART2A* summarization procedure, presentation of the modified update rule for the Batch SOM algorithm, used to generate the offline SOM

models, and formalization of the *average quantization error* assessment metric. Analysis of the StreamART2A algorithm and its parameters is performed in Section 4.4. In Section 4.5, evaluation of the overall methodology is performed, including a parameter sensitivity analysis for the StreamART2A algorithm and evaluation of obtained *offline* SOM models for exploratory cluster analysis. Section 4.5.6 provides an empirical validation of the introduced *average quantization error*. Finally, some final remarks and future work are discussed in Section 4.6.

## 4.2 Motivation and Aim

Motivated by the popular way of addressing data streams using a two-phase approach (reviewed in Section 2.6), one of the outlined research paths was to devise such a methodology adapted for offline exploratory cluster analysis by the SOM. With the intent to keep proposals in the ANN domain, the ART2-A algorithm (see Section 2.4.1.2) exhibits incremental learning properties — through the dynamic creation of categories representing a group of observations (recall Figure 2.7c), that is interesting to derive an online summarization procedure. However, as previously discussed, the vigilance parameter is hard to estimate and plays a critical role in the number and granularity of categories generated: if too high, it leads to the proliferation of fine categories; if too low, coarser categories are generated that may not capture a more detailed abstraction of the underlying distribution. ART networks also do not forget old information, one of the required characteristics for data streams regarding the non-stationarity problem.

Hence, the main aim is to develop an *online* abstraction methodology based on the ART2-A algorithm that can be leveraged *offline* by the SOM. Another aim is to develop a method that can be used to assess the fitness of the abstraction to the underlying distribution of the data stream, by adapting the standard quantization error  $E_q(t)$  (see Section 2.4.6) to a stream setting. This can potentially allow the development of change detection mechanisms and therefore enable, e.g., storing snapshots of current abstractions for later cluster analysis.

## 4.3 A Two-Phase StreamART2A/SOM Methodology

In (Silva and Marques 2012) the proposed methodology was presented, using a constrained variant of the ART2-A algorithm (Carpenter, Grossberg, and Rosen 1991a) for the abstraction step of the data stream, called StreamART2A. Contrary to the common usage of ART networks to find cluster centers, the idea from the micro-clustering strategy of CluStream (Aggarwal et al. 2003) was followed, with the goal of generating representative prototypes (*micro-categories*) of the streaming observations to produce an abstraction model. From this model, an *offline* SOM model were generated and exploratory cluster analysis performed. The average quantization error metric for model assessment and change detection was initially proposed and applied in (Silva and Marques 2012; Silva,

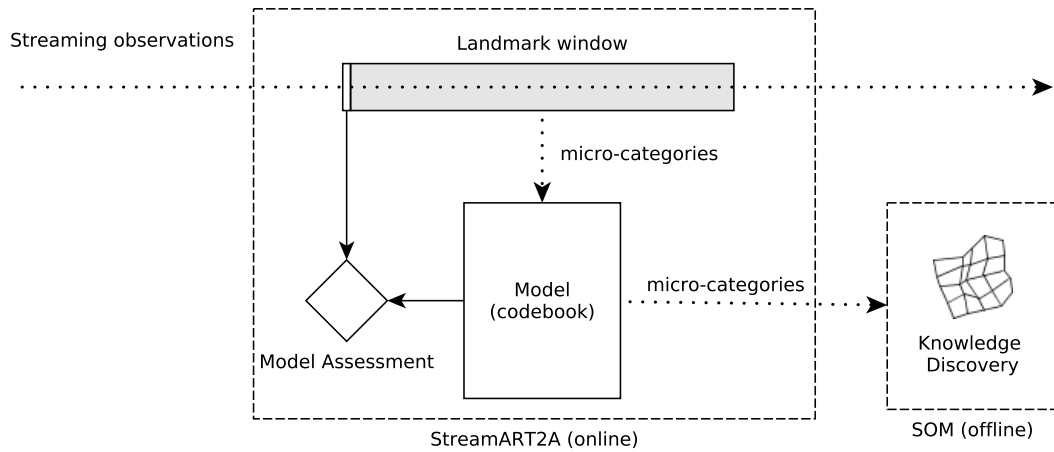


Figure 4.1: StreamART2A/SOM methodology overview.

Marques, and Panosso 2012).

#### 4.3.1 Methodology Overview

The proposed methodology is illustrated in Figure 4.1. The methodology consists in a two-phase approach to data streams: an online phase where the StreamART2A algorithm generates a fixed amount of *micro-categories* over a landmark window. There is an explicit differentiation between a category, i.e., a cluster center in ART terminology, and a micro-category, i.e., a representative prototype of a group of observations. The number of generated micro-categories should be much lower than the number of observations processed in the landmark window, effectively acting as a summarization procedure. The number of generated micro-categories per landmark window is kept fixed by the dynamic reduction of the vigilance parameter and a micro-category merge procedure. The idea of an ART network that restricts the number of categories has already been proposed in (Yin and Shen 2011). However, the proposed StreamART2A algorithm differs in many aspects, but share the *limit test* and vigilance update case. The generated micro-categories are maintained in a *first-in-first-out* fashion and correspond to the stored model, or *codebook*<sup>1</sup>.

Each micro-category contains a prototype of data, the number of observations it represents within the landmark window and a *timestamp*. After the processing of each landmark window, resulting micro-categories are added the stored model (codebook), which can be limited in its capacity. From the stored model, and leveraging the *timestamp* information, micro-categories from a specific and available elapsed interval of the learned data stream can be recovered and used *offline* as training observations by a Batch SOM algorithm (see Section 2.4.2.1) — with a modified update rule to take into account the observation counter of each micro-category. The *offline* exploratory cluster analysis is performed through the *U-Matrix* visualization (see Section 2.4.5).

<sup>1</sup>This terminology is often associated with a VQ procedure and with the SOM. The terminology is also applied to the model maintained by the StreamART2A algorithm for uniformity purposes.

For model assessment (and change indication) of the underlying data stream, the local quantization error is computed between the current observation being processed and the “closest” micro-category in the codebook. The sequence of these errors are averaged through a sliding window that can effectively indicate certain changes in the stream.

### 4.3.2 Notation

The StreamART2A operates over a multidimensional stream of continuous real-valued observations  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ , potentially unbounded ( $N \rightarrow \infty$ ), from a manifold  $\Omega \in \mathbb{R}^d$ . Each observation presented at time  $t$  is described by a  $d$ -dimensional feature vector, i.e.,  $\mathbf{x}(t) = [x_t^j]_{j=1}^d$ . The process is maintained over a landmark window (see Section 2.6.1.2) of size  $L$  where  $q$  micro-categories ( $q \ll L$ ) are generated. Each micro-category is a tuple  $Q_i = \langle \mathbf{w}_i, N_i^Q, T_i^Q \rangle$ , where  $\mathbf{w}_i$  is a prototype vector,  $N_i^Q$  is the number of observations assigned to the micro-category within the landmark window and  $T_i^Q$  is the *timestamp* indicating the create or last update time of the micro-category. Starting from an empty set  $\mathcal{Q}$  of micro-categories ( $m = 0$ ) and  $\rho = 1$ , the final set is obtained through dynamic reduction of  $\rho$ , which is a measure of similarity, and a micro-category merge procedure which limits their amount by the  $q$  threshold, i.e.,  $m \leq q$ . After each landmark window the resulting  $q$  micro-categories are added to the set  $\mathcal{K}$  (the stored model or codebook) and older ones removed. The codebook size is bounded by a parameter  $K$ .

### 4.3.3 The StreamART2A Algorithm

Given the assumed normalization of the input space (discussed in Section 3.5), the relation between distance and similarity of two vectors  $\mathbf{A}$  and  $\mathbf{B}$  is defined by:

$$dist(\mathbf{A}, \mathbf{B}) = \frac{\|\mathbf{A} - \mathbf{B}\|}{|\Omega|} \quad (4.1)$$

$$similarity(\mathbf{A}, \mathbf{B}) = 1 - dist(\mathbf{A}, \mathbf{B}) \quad (4.2)$$

Each streaming observation  $\mathbf{x}(t)$ , presented at time  $t$ , is compared to the  $m$  current prototypes in a *search* stage and the most similar prototype  $\mathbf{w}_c(t)$  is found. A *vigilance test*, defined in Eq. (4.3), is performed to verify if the degree of similarity between the example and most similar prototype  $\mathbf{w}_c$  is at least as high as the vigilance parameter  $\rho$ .

$$match = \begin{cases} true & \text{if } similarity(\mathbf{x}(t), \mathbf{w}_c(t)) \geq \rho \\ false & \text{otherwise} \end{cases} \quad (4.3)$$

If so, a *match* is found — the observation lies within the perceptive field of the micro-category, and the corresponding micro-category  $Q_c$  is chosen to represent the observation. Following a WTA strategy, the prototype  $\mathbf{w}_c(t)$  is updated by Eq. (4.4), where  $\eta$  is a constant *learning rate*, the value  $N_i^Q$  is incremented and the *timestamp* is set to the current



Notation	Description
$d$	data stream <i>dimensionality</i>
$t$	<i>time</i> , i.e., iteration number
$\mathbf{x}(t)$	<i>observation</i> presented at time $t$ , where $\mathbf{x}(t) \in \mathbb{R}^d$
$L$	size of landmark window
$q$	maximum number of micro-categories within a landmark window
$m$	number of micro-categories generated in the current landmark window
$Q_i$	a micro-category — a tuple $\langle \mathbf{w}_i, N_i^Q, T_i^Q \rangle$
$\mathbf{w}_i$	prototype vector of micro-category $Q_i$
$N_i^Q$	number of observations represented by $Q_i$
$T_i^Q$	<i>timestamp</i> of $Q_i$ (create or last update time)
$\mathcal{Q}$	set of all micro-categories per landmark window, bounded by $q$
$\mathcal{K}$	set of all stored micro-categories (codebook), bounded by $K$
$K$	maximum number of micro-categories to hold in the codebook
$E_q^l(t)$	normalized local quantization error for $\mathbf{x}(t)$
$ \Omega $	normalized input space manifold diagonal (see Section 3.5)
$\overline{qe}(t)$	average quantization error at time $t$
$T$	size of sliding window for $\overline{qe}(t)$ computation

Table 4.1: Notation for the StreamART2A algorithm.

time, i.e.,  $T_c^Q = t$ .

$$\mathbf{w}_c(t+1) = \eta \mathbf{x}(t) + (1 - \eta) \mathbf{w}_c(t) \quad (4.4)$$

Otherwise, a new micro-category is created, where the current observation is used as the prototype initialization. Until now, this follows the original ART2-A algorithm in its essence.

To limit the number of micro-categories created to  $q$ , a *limit test* is performed before processing the example. If the number of current micro-categories  $m$  exceeds  $q$ , the two most similar micro-categories are merged. Let

$$\{Q_i, Q_j\} = \max \{ \text{similarity}(\mathbf{w}_i, \mathbf{w}_j) : i, j = \{1, \dots, m\}, i \neq j \} \quad (4.5)$$

be the most similar pair of micro-categories. A new micro-category  $Q_{\text{merge}}$  is created with the prototype given by Eq. (4.6) and corresponding tuple by Eq. (4.7).

$$\mathbf{w}_{\text{merge}} = \left( \frac{N_i^Q}{N_i^Q + N_j^Q} \right) \mathbf{w}_i + \left( \frac{N_j^Q}{N_i^Q + N_j^Q} \right) \mathbf{w}_j \quad (4.6)$$

$$Q_{\text{merge}} = \langle \mathbf{w}_{\text{merge}}, N_i^Q + N_j^Q, t \rangle. \quad (4.7)$$

The vigilance parameter is dynamically updated according to Eq. (4.8) *only if* the similarity between the prototypes of the merged micro-categories is lower than the current vigilance value.

$$\rho^{\text{new}} = \begin{cases} \text{similarity}(\mathbf{w}_i, \mathbf{w}_j) & \text{if } \text{similarity}(\mathbf{w}_i, \mathbf{w}_j) < \rho \\ \rho & \text{otherwise} \end{cases} \quad (4.8)$$

Finally, the micro-categories  $Q_i$  and  $Q_j$  are discarded from the set  $\mathcal{Q}$  and  $Q_{\text{merge}}$  is added. Consequently, after a landmark window is processed we obtain the set  $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_q\}$ .

The StreamART2A model (or codebook)  $\mathcal{K}$  of micro-categories is maintained by this procedure: at the end of each landmark window the resulting  $q$  micro-categories are added to  $\mathcal{K}$  and the oldest removed. The maximum size of this codebook is defined by a user-defined parameter  $K$ , which is ultimately limited by the available memory or secondary storage space. Hence, the codebook effectively contains the last  $K$  generated micro-categories. The entire procedure is formalized in Algorithm 2. Optionally, obtained micro-categories that represent only one observation, or below some threshold, can be discarded in an attempt to deal with noisy observations. However, this study is reserved for future work and discussed in Section 4.6.

**Algorithm 2:** The StreamART2A algorithm.**Input:** $X$ : A sequence of observations  $\mathbf{x}(t)$  $L$ : Size of landmark window $q$ : Maximum number of micro-categories per landmark window $\eta$ : Learning rate used for updating a micro-category prototype $K$ : Maximum number of micro-categories to hold in the model**Output:** A codebook  $\mathcal{K}$ , containing  $K$  micro-categories

```

1 begin
2    $\mathcal{Q} \leftarrow \emptyset$ 
3    $\mathcal{K} \leftarrow \emptyset$ 
4    $m \leftarrow 0$ 
5    $\rho \leftarrow 1$ 
6    $t \leftarrow 0$ 
7   foreach  $\mathbf{x}_t$  do
8     if  $m > q$  then // limit test
9       Find most similar micro-categories  $\{Q_i, Q_j\}$  using Eq. (4.5)
10      Create  $Q_{merge}$  using Eq. (4.7)
11       $s \leftarrow \text{similarity}(\mathbf{w}_i, \mathbf{w}_j)$ 
12      if  $s < \rho$  then
13         $\rho \leftarrow s$ 
14      end
15       $\mathcal{Q} \leftarrow \mathcal{Q} - \{Q_i, Q_j\}$ 
16       $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_{merge}\}$ 
17    end
18    Find  $\mathbf{w}_c(t)$  and perform vigilance test using Eq. (4.3)
19     $s \leftarrow \text{similarity}(\mathbf{x}(t), \mathbf{w}_c(t))$ 
20     $match \leftarrow false$ 
21    if  $s \geq \rho$  then // vigilance test
22       $match \leftarrow true$ 
23    end
24    if  $match = true$  then // update micro-category  $Q_c$ 
25       $\mathbf{w}_c(t+1) = \eta \mathbf{x}(t) + (1 - \eta) \mathbf{w}_c(t)$ 
26       $N_c^Q \leftarrow N_c^Q + 1$ 
27       $T_c^Q \leftarrow t$ 
28    else // create new micro-category
29       $Q_{new} \leftarrow \langle \mathbf{x}_t, 1, t \rangle$ 
30       $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_{new}\}$ 
31    end
32     $t \leftarrow t + 1$ 
33    if  $t \bmod L = 0$  then // check end of landmark window
34       $\mathcal{K} \leftarrow \mathcal{K} \cup \mathcal{Q}$ 
35       $\mathcal{Q} \leftarrow \emptyset$ 
36       $m \leftarrow 0$ 
37       $\rho \leftarrow 1$ 
38    end
39  end
40 end

```

One consequence of this process is that during the first  $L$  observations of the data stream the model is effectively empty. Also, when processing a landmark window, their respective micro-categories are not yet part of the codebook. While we can also use the current  $m$  micro-categories as part of the model, there is no guarantee that they had the opportunity to converge to truly representative prototypes. However, misrepresentations should be diluted in a bigger pool of micro-categories.

The maximum time horizon ( $h_t$ ) permitted by the codebook is given by Eq. (4.9). Consequently, it is possible to recover micro-categories that are within the interval  $[t - h_t, t]$  — if additionally using the micro-categories contained in the landmark window (see above).

$$h_t = L \frac{K}{q} \quad (4.9)$$

Consequently, evolving data, i.e., non-stationarity of the data stream, is dealt by an implicit sliding window imposed by  $K$ , where only the more recent micro-categories are retained. This strategy is similar to ones of current surveyed methods in Section 2.6.1.

#### 4.3.4 Modified Batch SOM Algorithm

From the maintained codebook of the StreamART2A algorithm, micro-categories for a specific elapsed interval  $[t_1, t_2]$ , with  $t_1 \ll t_2$ , of the data stream can be recovered to compose a dataset  $\mathcal{D} = \{Q_i : T_i^Q \in [t_1, t_2]\}$ , which is used to generate an *offline* Batch SOM model from where to perform exploratory cluster analysis on the underlying distribution during the period  $[t_1, t_2]$ . The Batch SOM variant was chosen because it is faster, has one less parameter and we do have all necessary input patterns. Consequently, each input pattern presented to the Batch SOM algorithm is a prototype from a micro-category in the dataset, i.e.,  $\mathbf{x}(t) \equiv \mathbf{w}_i \in \mathcal{D}$ . We shall refer to these particular input patterns as *input prototypes*.

A modification to the update rule, enabling it to account the “weight” of an input prototype, is introduced. Only by this way can we keep the input space density matching in the SOM projection. Equations (4.10) and (4.11) formalize the weight computation and modified Batch SOM update rule, respectively.

$$\Delta \mathbf{x}(t) = \Delta \mathbf{w}_i = \frac{N_i^Q}{\max\{N_j^Q : Q_j \in \mathcal{D}\}} \quad (4.10)$$

$$\mathbf{w}_k(t_f) = \frac{\sum_{t'=t_0}^{t'=t_f} \Delta \mathbf{x}(t) h_{ck}(t') \mathbf{x}(t)}{\sum_{t'=t_0}^{t'=t_f} \Delta \mathbf{x}(t) h_{ck}(t')} \quad (4.11)$$

The weight of an input prototype is the ratio between the number of observations the

respective micro-category represents and the maximum number of observations represented by any other micro-category in  $\mathcal{D}$ . Therefore,  $\Delta \mathbf{x}(t) \in ]0, 1]$ . This modified Batch SOM update rule is similar to the original — see Eq. (2.6), but weighs the contribution of each input prototype in the self-organization process.

A direct benefit of this methodology is that by comparing models at different intervals, cluster evolution analysis is possible.

#### 4.3.5 Trend of Model Adaptation by the Average Quantization Error

Since  $N \rightarrow \infty$ , the quantization error (QE) measure was adapted to data streams as a moving average<sup>2</sup> and denominated *average quantization error*, denoted by  $\overline{qe}$ , based on the premise that the error of a learner will decrease over time for an increasing number of examples if the underlying distribution is stationary; otherwise, if the distribution changes, the error increases (Gama et al. 2004). Ultimately, it gives us the trend of the codebook adaptation to the underlying stream. Afterwards, change detection mechanisms can be attempted over  $\overline{qe}$ .

Hence, for each incoming observation  $\mathbf{x}(t)$  we find the prototype  $\mathbf{w}_c(t) \in \mathcal{K}$  that is closest to  $\mathbf{x}(t)$  and compute the normalized local quantization error  $E'_q(\mathbf{x}_t)$  according to Eq. (4.12). These values are averaged over a window of length  $T \gg 1$  to obtain the average quantization error  $\overline{qe}(t)$ , as in Eq. (4.13). It should be obvious that the first  $\overline{qe}(t)$  is only available at  $t = T + 1$ , starting from  $t = 0$ .

$$E'_q(t) = \text{dist}(\mathbf{x}(t), \mathbf{w}_c(t)) = \frac{\|\mathbf{x}(t) - \mathbf{w}_c(t)\|}{|\Omega|} \quad (4.12)$$

$$\overline{qe}(t) = \frac{1}{T} \sum_t^{t-T+1} E'_q(t) \quad (4.13)$$

Consequently, the value of  $T$  establishes a short, medium or long-term trend of the model adaptation, but is bounded by  $K$ .

## 4.4 StreamART2A Algorithm Analysis

The StreamART2A algorithm provides a summarization of the data stream, in the form of micro-categories, through a landmark window. One could be led to believe that the observations are processed in batches of  $L$  observations, but this is not the case. Observations are processed sequentially and no past observations are kept in memory by the algorithm.

An analysis regarding the processing within each landmark window should be made. The ratio  $L/q$  can be seen as the *summarization ratio*, e.g., for  $L = 1000$  and  $q = 20$  this

<sup>2</sup>In signal processing a moving average can be seen as a low-pass filter. Low-pass filters provide a smoother form of a signal, removing the short-term fluctuations, and leaving the longer-term trend.

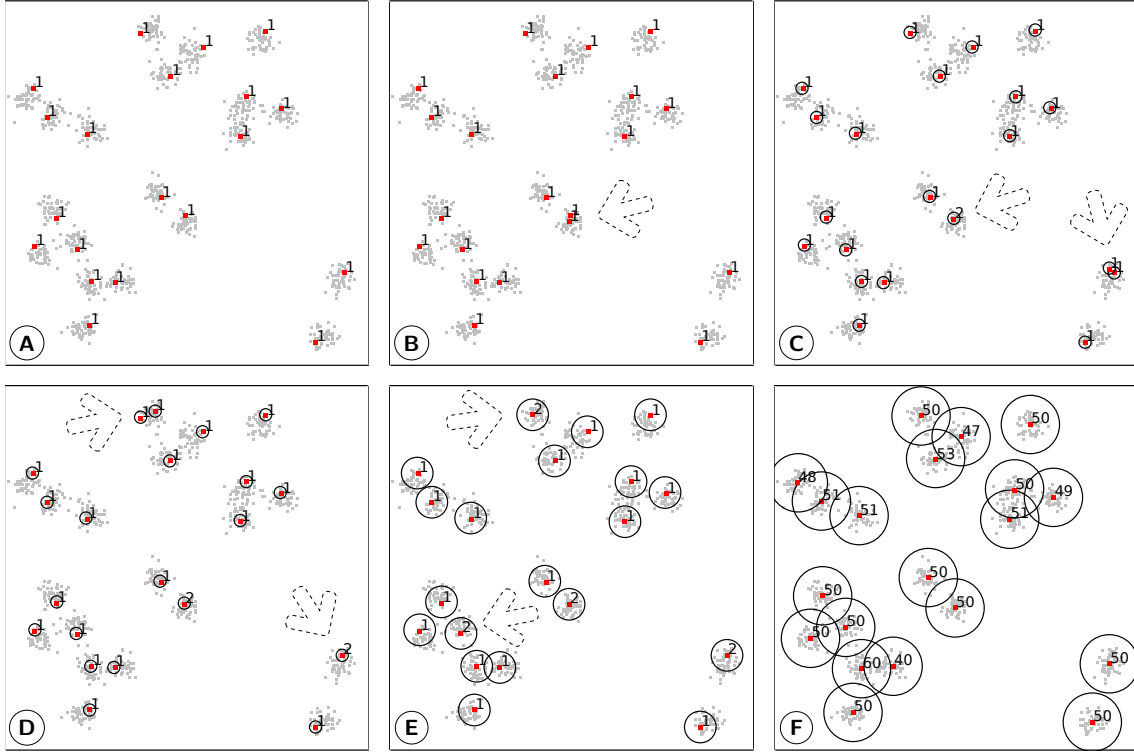


Figure 4.2: StreamART2A: example of procedures in a landmark window, where  $q$  equal to the number of underlying clusters.

gives us a ratio of 50. In this concrete case, and as an example, a micro-category can represent an average of 50 observations if these are drawn from a well defined 20 Gaussian cluster structure with equal probability; the prototypes adjust to the centers of the clusters by gradient descend and the perceptive field established by  $\rho$  covers each cluster. This example is illustrated in a two-dimensional case in Figure 4.2, showing the resulting micro-categories within the first landmark window, after specific iterations (starting from  $t = 0$ ): (A)  $t = 19$ , (B)  $t = 20$ , (B)  $t = 21$ , (B)  $t = 22$ , (B)  $t = 23$  and (B)  $t = 999$ . In (A) there is one micro-category assigned to an observation of each cluster. In (B) a new micro-category is created, totaling  $m = q + 1$  micro-categories. In (C) the *limit test* triggers a merge procedure, where the two identified micro-categories are merged and  $\rho$  is decreased (resulting  $m = q$ ); the observation processed in this iteration leads to the creation of a new micro-category (again,  $m = q + 1$ ). In (D) a similar processing to (C) occurs, but in this iteration  $\rho$  is not updated — the similarity between those two micro-categories is at least equal to the vigilance. In (E) another two micro-categories are merged and  $\rho$  updated ( $m = q$ ); the current observation is assigned to an existing micro-category. This endures until the end of the landmark window, where (F) depicts the resulting  $q$  micro-categories.

Although the previous parameterization coped well with the data, real data is not this well-behaved. If just analyzing the StreamART2A algorithm over one landmark window,

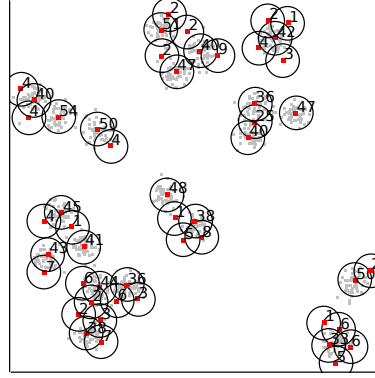


Figure 4.3: StreamART2A: alternate result regarding Figure 4.2, with  $q$  greater than the number of underlying clusters.

it can be deemed unstable, because the micro-categories that are generated are dependent upon the order of the observations. In the previous example, if the first  $q$  observations were not drawn uniformly from all clusters, the consequence would be that some micro-categories would represent more than one cluster and their centers may not lie over any input data. Consequently, a value of  $q$  smaller than the total number of underlying clusters forces one micro-category to represent more than one cluster.

However, the purpose of the StreamART2A algorithm is not to produce categories representing clusters, but micro-categories, in the sense that the goal is for a micro-category to represent a group of observations, not necessarily a cluster. The cluster analysis is later performed by the SOM algorithm. Figure 4.3 presents the result with  $L = 1000$ , but  $q = 50$ , in a situation where the observations are not drawn uniformly from the natural clusters. We can see that each cluster is being represented by more than one micro-category, effectively abstracting the underlying cluster structure. It is expected that micro-categories from subsequent landmark windows have different prototypes. Thus, a set of several landmark summarizations used as input prototypes should allow the SOM to converge to the natural clusters. Other examples of resulting landmark windows over different data streams are later provided in Section 4.5.4.

As a final observation regarding the value  $K$ , besides establishing the maximum micro-categories in the codebook, it should at least allow the recovery of sufficient micro-categories to train a SOM of a given size; in this case the dataset will be relatively small (see Section 2.4.4).

#### 4.4.1 Time and Space Complexity

Before discussing the time and space complexity of the algorithm, we should discuss the storage mechanism of the codebook, which can impose an additional computational overhead, dependent of the parameters  $K$  and  $q$ . If  $K$  is a multiple of  $q$ , then the codebook

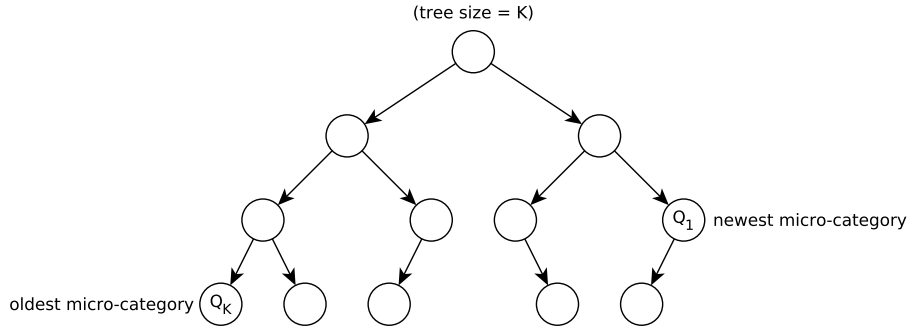


Figure 4.4: A StreamART2A codebook implemented by means of a balanced binary tree.

can be implemented as a simple *queue* with a fixed size. However, if it  $K$  is not a multiple of  $q$ , the most efficient way (regarding access/insert/delete operations) to ensure that only the last  $K$  micro-categories are kept is by using a binary tree, with the timestamp as a node comparator, as shown in Figure 4.4. However, this imposes an additional overhead at the end of each landmark window, since the insert and delete operations in a binary tree have an average complexity of  $O(\log n)$  and  $O(n)$  in the worst case (if not balanced), contrary to the linear complexity offered by an efficient implementation of a queue.

Consequently, establishing  $K$  as a multiple of  $q$  is recommended. The time complexity analysis will assume a queue is used.

The processing of each observation has a time complexity of  $O((qd)^2 + qd)$  in the worst case, i.e., one merge procedure and the search stage; a micro-category create/update procedure is performed in  $O(1)$ . However, it is expected that most observations within the landmark window, with decreasing  $\rho$ , do not trigger a merge between micro-categories. At the end of each landmark window there is an overhead of  $O(q)$  to move the generated  $q$  micro-categories to the codebook  $\mathcal{K}$ . Notwithstanding the variability in the number of actual operations needed per observation, the StreamART2A algorithm processes the observations in constant average time. Regarding space, the StreamART2A imposes a total space complexity of  $O(qd + Kd)$ , i.e., the memory needed to hold the micro-categories within the current landmark window and the micro-categories in the codebook. The computation of the average quantization error imposes additional time and space constraints upon the StreamART2A algorithm: each observation requires additional  $O(Td)$  time and space is increased by  $O(T)$  to store the last  $T$  values (assuming an efficient queue is used).

## 4.5 Experimental Evaluation

A series of experiments was conducted to evaluate the proposed methodology using stationary and non-stationary artificial data streams. Artificial data was chosen for this purpose so we can establish the ground truth of the expected outcome and illustrate some key points.



Name	$d$	$N$	Data Stream			Type of Evaluation			
			Stationary	Change at	#Clusters	PSA	VIL	ECA	MA
<i>Gauss</i>	2	100 000	Y	-	1	+	✓	✓	
<i>Complex</i>	2	100 000	Y	-	7	✓	✓	✓	✓
<i>Chain</i>	3	100 000	Y	-	2	+	✓	✓	
<i>d2k20</i>	2	300 000	Y	-	20	+	✓		
<i>d3k20</i>	3	300 000	Y	-	20	+	✓		
<i>d5k20</i>	5	300 000	Y	-	20	✓	✓	✓	
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2	✓	✓		✓
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2	✓	✓	✓	✓
<i>Hepta</i>	3	150 000	N	100 001	7/6	+	✓	✓	✓

Table 4.2: Experimental evaluation overview for the StreamART2A/SOM methodology. The symbol ✓ identifies results presented in this chapter; experiments marked with + are presented in Appendix C. Parameter sensitivity analysis (PSA); Vigilance impact and landmark examples (VIL); exploratory cluster analysis (ECA), and; model assessment and change indication (MA).

#### 4.5.1 Overview

The presented experiments focus on the following evaluations:

- i. Parameter sensitivity analysis (PSA) for the StreamART2A algorithm to assess the impact of the parameters  $\{L, q, \eta\}$  in the summarization procedure. This is presented in Section 4.5.3 and the objective is to derived good empirical all-around parameters from the tested data streams.
- ii. Obtained vigilance  $\rho$  values and comparison of micro-categories generated within a landmark window for different parameterizations — presented in Section 4.5.4.
- iii. Offline exploratory cluster analysis by Batch SOM models generated from micro-categories extracted at specific time-intervals of the data streams; the visually inferred clusters are compared with the expected outcome — presented in Section 4.5.5 and aims at evaluating the presented methodology.
- iv. Change indication in the underlying data streams through the use of the  $\overline{qe}(t)$  model assessment metric — presented in Section 4.5.6.

Table 4.2 summarizes the artificial data streams used in each type of evaluation previously described. This table also identifies some additional results that can be found in Appendix C, namely in Section C.1.

### 4.5.2 Data Normalization and Algorithm Parameters

All artificial data streams were normalized in the unit hypercube, i.e.,  $\mathbf{x}(t) \in [0, 1]^d$ . The StreamART2A algorithm parameters are derived after the PSA analysis of the following section. The parameterization of the Batch SOM algorithm is performed empirically, since only “rules-of-thumb” exist towards finding good parameters (Kohonen 2001). Concerning the lattice size it should be rectangular in order to minimize projection distortions, hence we use a  $20 \times 40$  lattice. The remaining parameters values were  $\eta_i = 0.1$ ,  $\eta_f = 0.01$ ,  $\sigma_i = 1/2\sqrt{20^2 + 40^2}$  and  $\sigma_f = 1$  with training enduring 10 and 40 ordering and convergence epochs, respectively. Justification for the choice of these parameters can be found in Section 2.4.4.

### 4.5.3 Parameter Sensitivity Analysis

This section presents a parameter sensitivity analysis (PSA) for the StreamART2A algorithm to assess the impact of the parameters  $\{L, q, \eta\}$  in the summarization procedure, regarding the relative quantization errors of resulting micro-categories and the mean number of merges in the landmark windows. The parameter  $L$  determines the size of the landmark window,  $q$  establishes the maximum number of micro-categories to generate for each landmark window and  $\eta$  dictates the gradient descend step size adjustment of a micro-category prototype update, i.e., the *learning rate*.

The parameter sensitivity analysis was performed in three dimensions with  $L = \{500, 1000, 1500, 2000\}$ ,  $q = \{20, 50, 100, 200\}$  and  $\eta = \{0.05, 0.1, 0.2, 0.5\}$  as the parameter-space, against the following two statistics:

**Mean quantization error.** After a landmark is processed and  $q$  micro-categories are generated, the presented  $L$  observations are reused to compute the mean quantization error of the landmark window, i.e.,  $\left( \sum_{i=t}^{t+L-1} E'_q(t) \right) / L$ , being  $E'_q(t)$  computed with  $\mathbf{w}_c(t)$  inside the landmark window. The final obtained statistic ( $\text{Mean } E'_q(t)$ ) is the mean value obtained between all landmark windows until the end of the data stream.

**Mean number of merges.** Computes the mean number of merges across all landmark windows, i.e., for each landmark window the number of micro-category merges is accumulated and averaged between the total number of landmark windows used throughout the data stream.

Figure 4.5 presents the results of the PSA analysis for the *Complex* and *d5k20* stationary data streams. Analogously, Figure 4.6 presents the results for the *AbruptOneTwo* and *Clouds* non-stationary data streams. To preserve the flow of this chapter, results for the other data streams can be found in Section C.1.

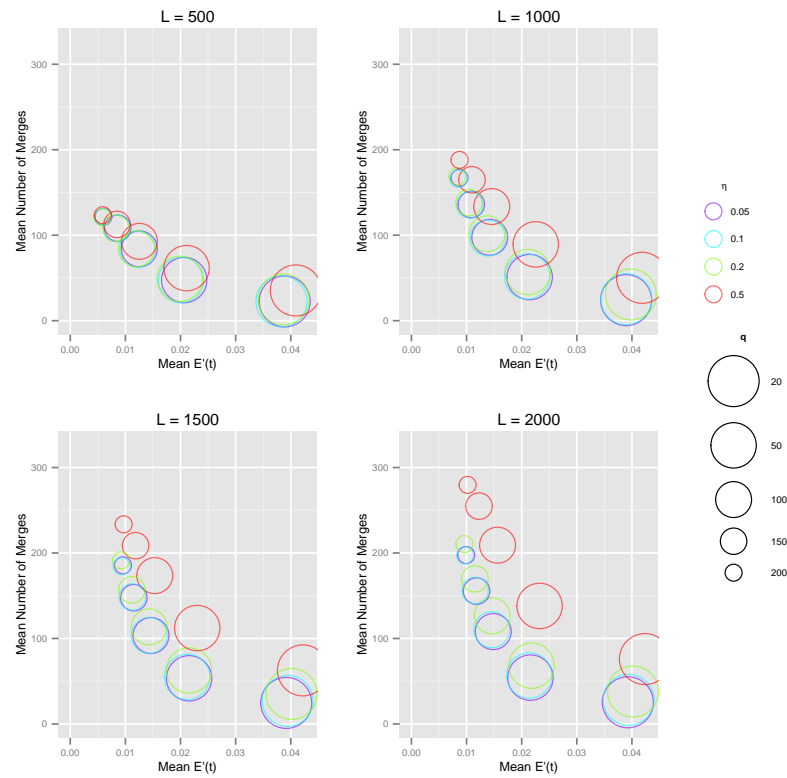
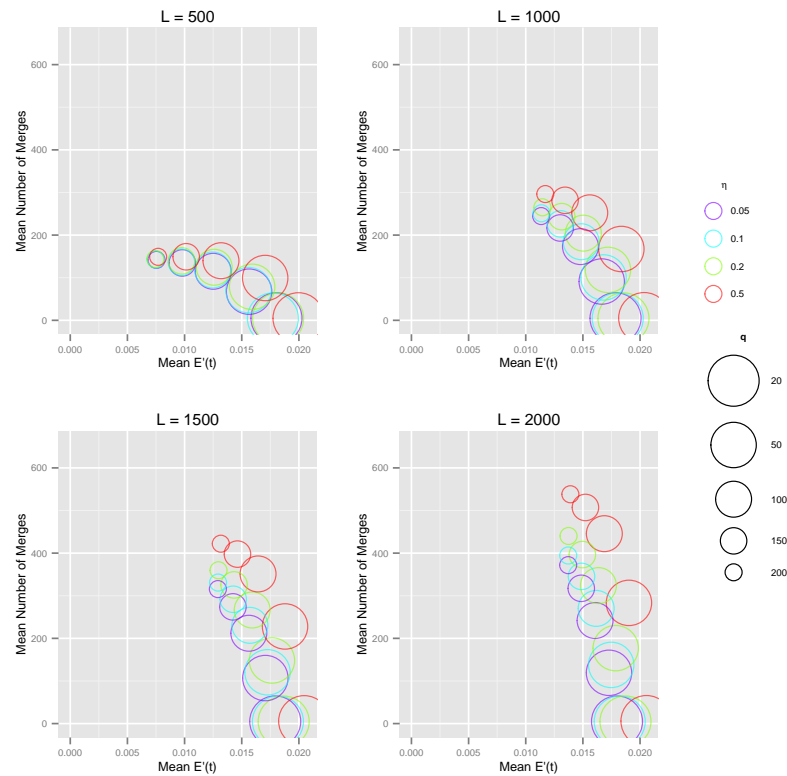
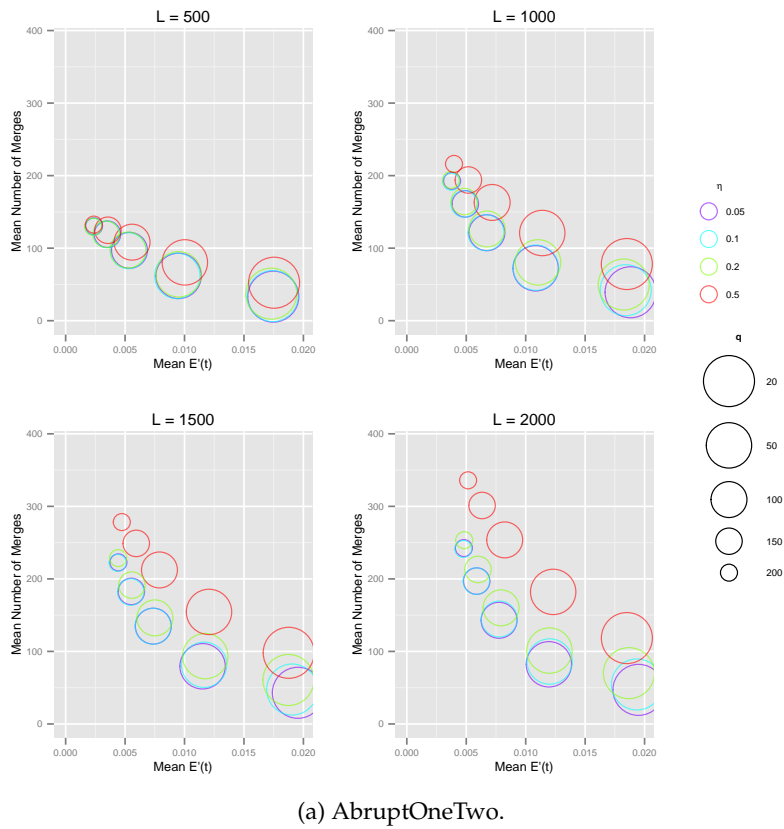
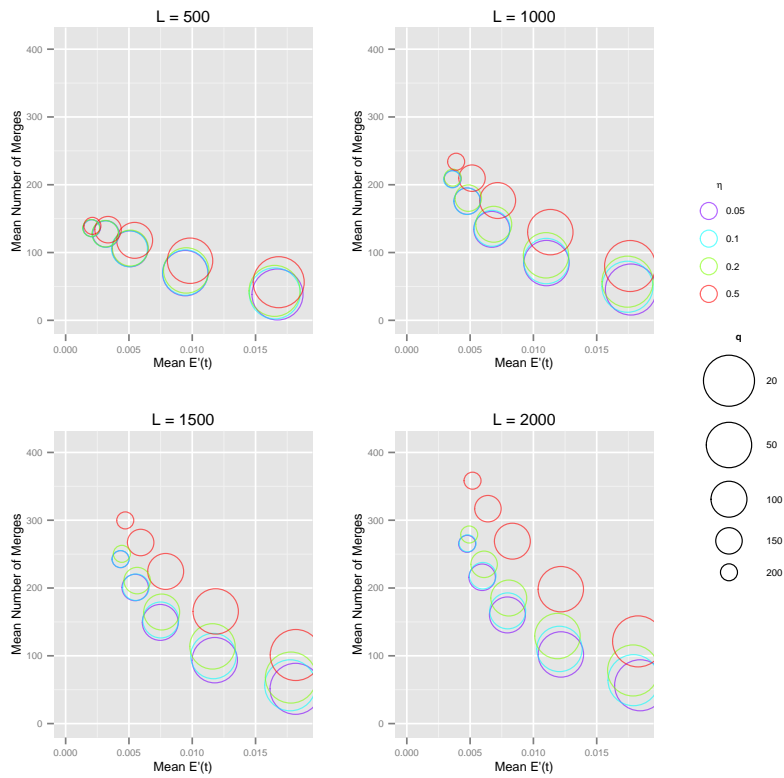
(a) *Complex* data stream.(b) *d5k20* data stream.

Figure 4.5: Parameter sensitivity analysis results for the StreamART2A algorithm over stationary data streams.



(a) AbruptOneTwo.



(b) Clouds.

Figure 4.6: Parameter sensitivity analysis results for the StreamART2A algorithm over non-stationary data streams.

We are interested in the best balance between a low  $\text{Mean } E'_q(t)$  and the minimum mean number of merges, since the later penalizes the computational cost of a landmark window. There is also a trade-off to achieve regarding the summarization ratio  $L/q$ , i.e., although it should be obvious that a lower ratio yields better results (less observations to represent by the same number of micro-categories), we are interested in summarizing the higher number of observations by the least amount of micro-categories to keep the model more compact.

First, regarding the learning rate  $\eta$ , it becomes clear that the value 0.5 yields the worst results. This is expected, since a value this high induces more pronounced updates in the prototypes, preventing them to converge steadily to the center of the represented observations. As for the other values, results are similar, but  $\eta = 0.05$  seems to globally attain better results independently of the other parameters.

As for the parameter  $q$ , the value  $q = 50$  seems to give an overall best balance between a lower mean quantization error and lower mean number of merges across the tested data streams. Higher values of  $q$  naturally achieve a best quantization procedure, but the smallest perceptive fields induce a higher number of merges. The contrary happens with  $q = 20$ , for example. Moreover, if we take into account the example provided in Section 4.4 regarding the possible number of underlying clusters, indeed  $q = 50$  seems like a better value.

Finally, in respect to the parameter  $L$ , and assuming  $\eta = 0.05$  and  $q = 50$ , a lower  $L$  value provides better results as expected. However, since the aim is to also obtain a good summarization ratio, the choice of  $L = 1000$  seems a good compromise. Also, pertaining the *offline* SOM models generation and corresponding lattice size of  $20 \times 40$ : by choosing this value we can obtain 1000 micro-categories for each 20 000 presented observations to the StreamART2A algorithm, which should be sufficient to generate a good SOM model for a time interval of that length. If we opted for  $L = 2000$ , then for such a time interval we would only obtain half of the micro-categories, i.e., 500. This highlights the fact that the choice of the parameters may be influenced by the size of the target SOM model to produce and granularity of time intervals of interest. Consequently, the recommended ground parameters derived from this PSA and discussion are  $L = 1000$ ,  $q = 50$  and  $\eta = 0.05$ .

Additionally, Figure 4.7 exhibits the behavior of the mean number of merges using the *d5k20* data stream. Naturally, the mean number of merges converges to the  $L$  asymptotic value as  $q$  increases.

A brief analysis of the impact of increasing dimensionality of the data stream is illustrated in Figures 4.8 and 4.9 regarding the mean number of merges and scalability of the algorithm, respectively. For this purpose, the *d2k20*, *d3k20* and *d5k20* data streams were used, since they describe a similar cluster structure in different dimensional spaces.

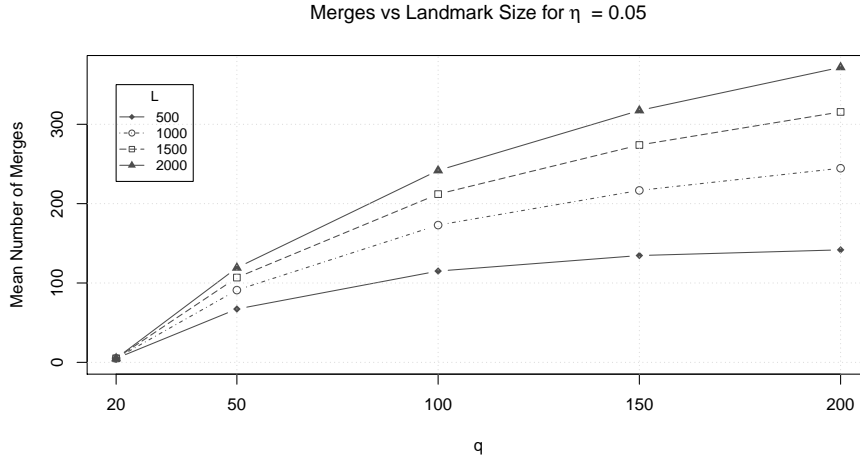


Figure 4.7: Impact of landmark window size  $L$ , with varying  $q$ , in mean number of merges over the *d5k20* data stream.

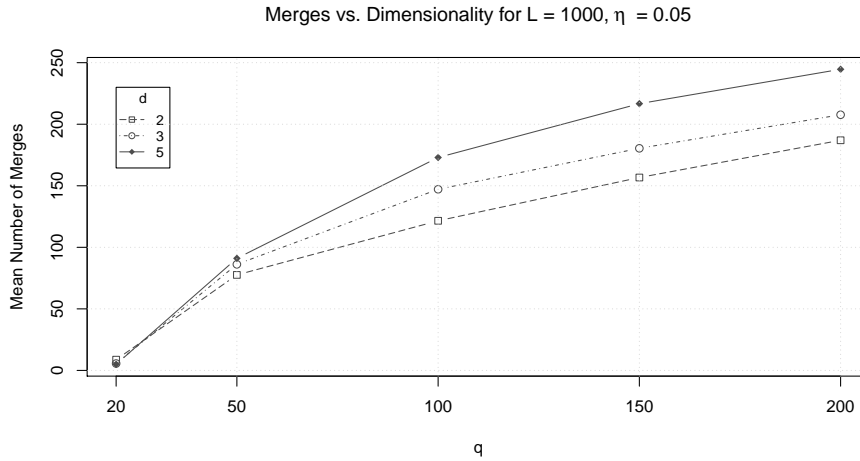


Figure 4.8: Impact of data dimensionality, with varying  $q$ , in mean number of merges.

In respect to the mean number of merges, they increase together with data dimensionality, given the relative higher distances encountered in higher dimensional spaces. As for the scalability of the summarization procedure, the cost increases due to the number of dimensions to consider in the distance computations; the impact of the parameter  $q$  is higher due to the higher number of merges necessary. The scalability regarding  $q$ , although not linear, is proportional to  $q$  (plus the overhead of the merges) which is in line with the algorithm analysis performed in Section 4.4.1.

#### 4.5.4 Vigilance and Landmark Window Illustrations

This section provides an evaluation and illustrations of the effect of the StreamART2A algorithm parameterization on the generated micro-categories within landmark windows.

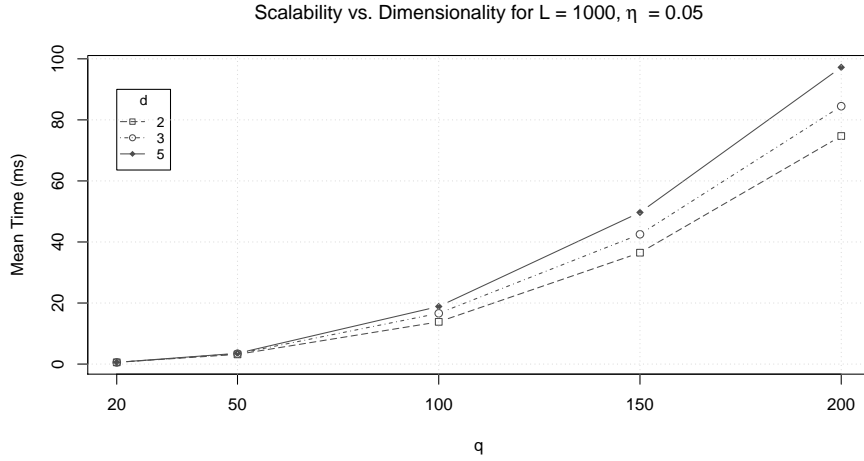


Figure 4.9: Impact of data dimensionality, with varying  $q$ , in the computational cost of a landmark window.

In particular, the  $q$  value has a direct impact on the vigilance values obtained within landmark windows, i.e., higher values produce “smaller” micro-categories regarding their perceptive fields and, by consequence, higher vigilance values. In the opposite direction, a lower number of micro-categories to represent the same amount of observations yields “larger” perceptive fields and lower vigilance values.

Table 4.3 presents the vigilance  $\rho$  values obtained for all used data streams in the present evaluation regarding the mean and standard deviation across all landmark windows. The ground parameters derived from the PSA were used, together with some variations for comparison and illustration purposes. Overall, we can see that the standard deviations value are very low, an indication that the summarization process is stable over stationary and non-stationary data streams. Also, for the same ground parameterization, the attained vigilance values vary for the different underlying cluster structures of the data streams. This is an indication that there is no fixed value that can be derived for all problems, further justifying the dynamic adjustment of the vigilance values within landmark windows. Moreover, the results confirm the relationship between the parameter  $q$  and the obtained vigilance values stated in the previous paragraph.

Concerning particular cases chosen to illustrate some key points:

- Figures 4.10a and 4.10b illustrate the resulting micro-categories for the first landmark window of the *Complex* data stream, varying the parameters  $L$  and  $q$ , but with the same summarization ratio. It is visible that using  $q = 100$  produces a finer abstraction of the cluster structure, but with a higher computational cost, i.e., a higher mean number of merges exhibited in the PSA. However, in practical terms we must recall that micro-categories are generated continuously for subsequent landmark windows, which allow a larger group of coarser micro-categories generated with  $q = 50$  to also describe the underlying cluster structure with success, e.g., subsequent prototypes will rarely be in the same positions. This is demonstrated in the

Data Stream			Parameters			$\rho$		
Name	$d$	Stationary	$L$	$q$	$\eta$	Mean	Std.	Illustration
<i>Gauss</i>	2	Y	1000	50	0.05	0.95309	0.00169	
<i>Complex</i>	2	Y	1000	50	0.05	0.95242	0.00119	Figure 4.10a
			2000	100	0.05	0.96852	0.00049	Figure 4.10b
<i>Chain</i>	3	Y	1000	50	0.05	0.95256	0.00097	Figure 4.11a
			2000	50	0.05	0.95241	0.00101	Figure 4.11b
<i>d2k20</i>	2	Y	1000	50	0.05	0.96913	0.00094	
<i>d3k20</i>	3	Y	1000	50	0.05	0.96882	0.00084	Figure 4.12a
			1000	20	0.05	0.94636	0.00426	Figure 4.12b
			1000	10	0.05	0.81622	0.02049	Figure 4.12a
<i>d5k20</i>	5	Y	1000	50	0.05	0.97116	0.00062	
<i>AbruptOneTwo</i>	2	N	1000	50	0.05	0.97686	0.00412	
<i>Clouds</i>	2	N	1000	50	0.05	0.97646	0.00145	
<i>Hepta</i>	3	N	1000	50	0.05	0.95669	0.00179	

Table 4.3: Obtained vigilance values across all tested data streams for various parameterizations.

following section when generating the offline SOM models;

- Figures 4.11a and 4.11b depict the first landmark window for the *Chain* data stream with  $L = 1000$  and  $L = 2000$ , respectively. Two things can be inferred: first, from the previous PSA (Figure 4.5) we can see that the later parameterization leads to more micro-category merges, but looking at the obtained vigilance values they are identical. From here we can infer that, with  $q = 50$ , in a stationary phase of the data stream, no more than 1000 observations are necessary to derive a good overall vigilance  $\rho$  value;
- In the StreamART2A algorithm analysis, a recommendation was made in relation to choosing values of  $q$  at least as high as the maximum number of expected clusters in the underlying distribution. The *d3k20* data stream describes a cluster structure composed by 20 clusters in three-dimensional input space. Figures 4.12a, 4.12b and 4.12c illustrate example micro-categories generated within a landmark window using  $q = \{50, 20, 10\}$ , respectively. While the larger values allow a correct summarization of the cluster structure, using  $q = 10$  generates micro-categories whose centers do not lie over the cluster centers. The consequence of this is further illustrated in the obtained offline SOM models of the next section.

Finally, using the same parameterization, the obtained vigilance values for the *d2k20*, *d3k20* and *d5k20* do not vary significantly. However, this is because the derived vigilance



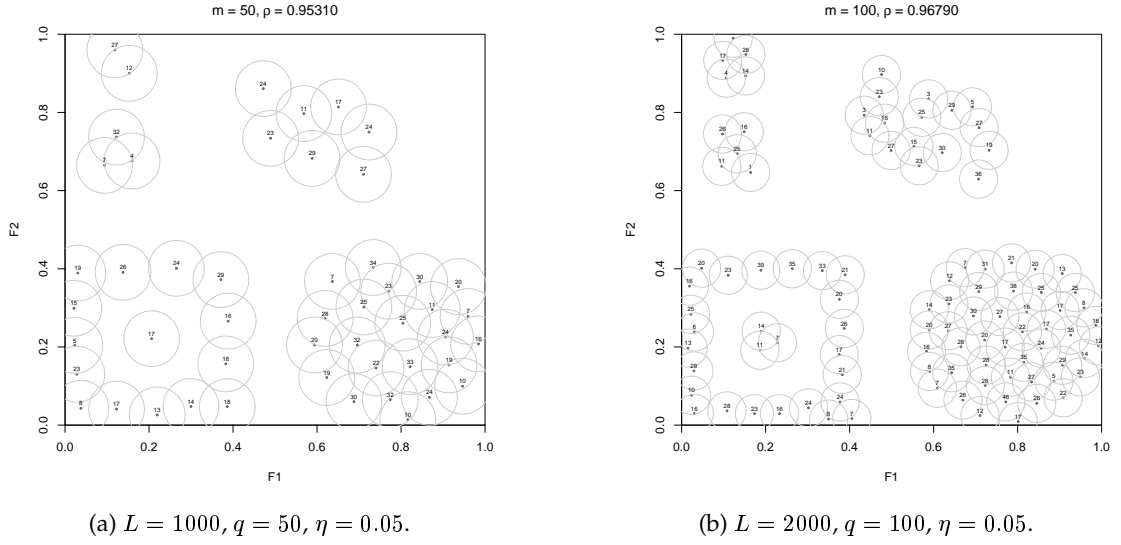


Figure 4.10: Generated micro-categories for first landmark window over the *Complex* data stream.

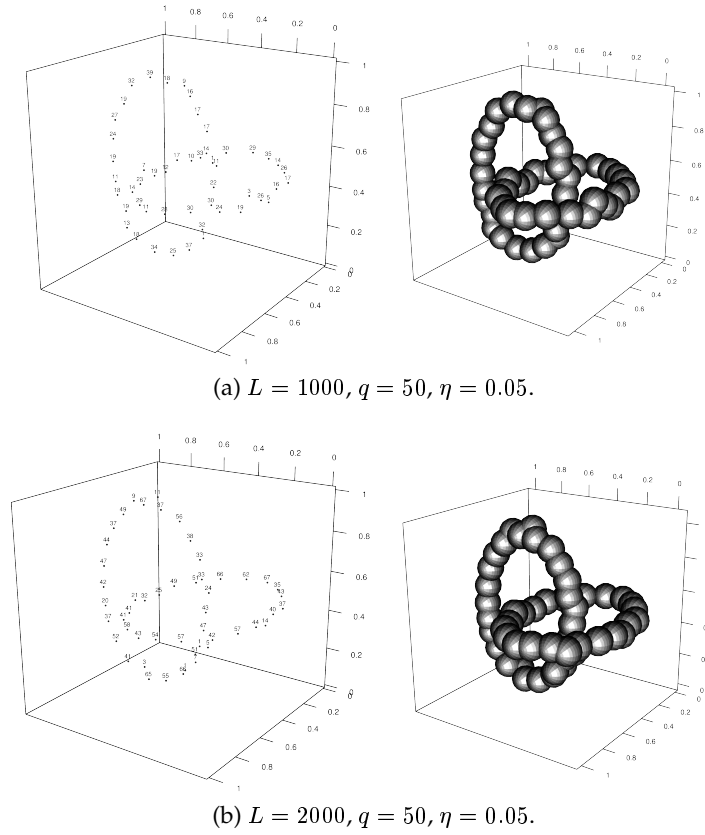
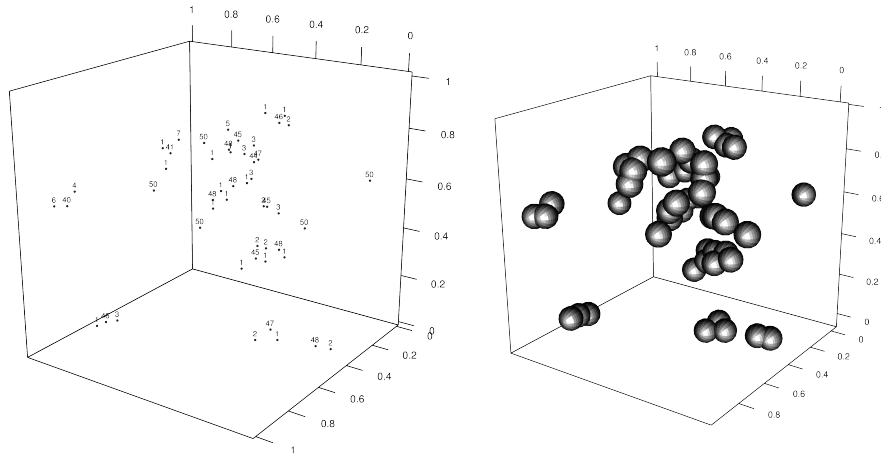
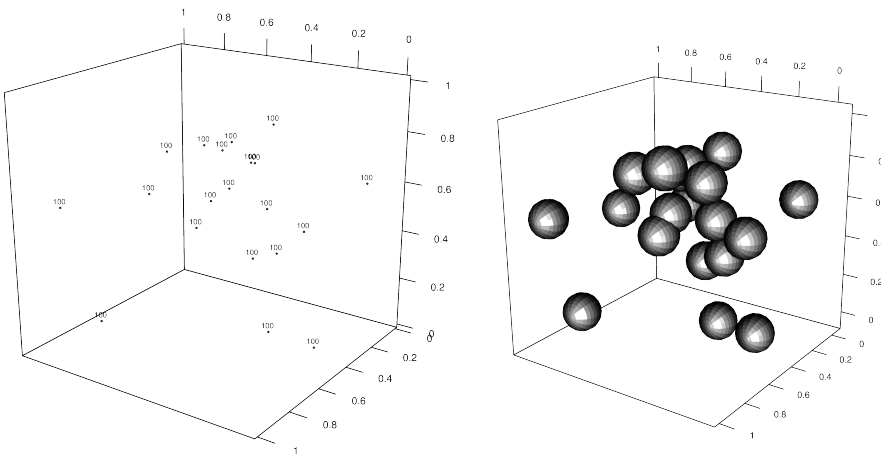
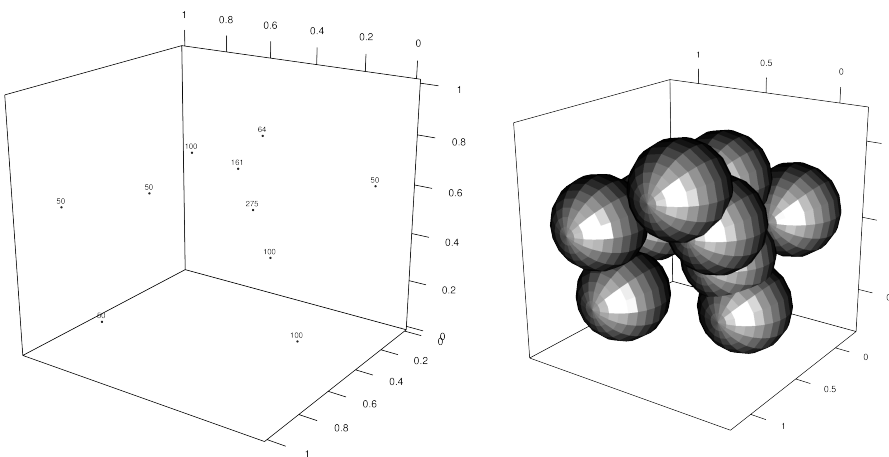


Figure 4.11: Generated micro-categories for first landmark window over the *Chain* data stream.

(a)  $L = 1000, q = 50, \eta = 0.05$ .(b)  $L = 1000, q = 20, \eta = 0.05$ .(c)  $L = 1000, q = 10, \eta = 0.05$ .Figure 4.12: Generated micro-categories for first landmark window over the *d3k20* data stream.

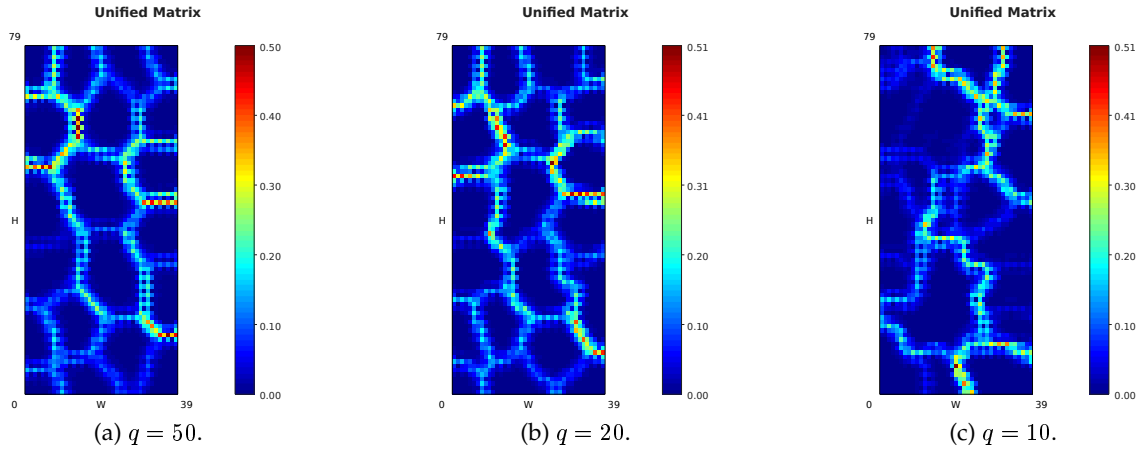


Figure 4.13: Obtained *d5k20* offline SOM models for  $t_1 = 0$ ,  $t_2 = 20\,000$ , with varying  $q$  values.

values are obtained with normalized distances — see Equations (4.1) and (4.2).

#### 4.5.5 Exploratory Cluster Analysis

In this section, *offline*  $20 \times 40$  Batch SOM models — using the new update rule presented in Eq. (4.11), for a time horizon of 20 000 observations are presented, from where exploratory cluster analysis is performed through the derived *U-Matrix* visualization. For such a time horizon, this yields a necessary value of  $K \geq 1000$  — Eq. (4.9). All results were obtained with parameters  $L = 1000$ ,  $q = 50$  and  $\eta = 0.05$ , except were stated. The parameterization of the Batch algorithm remains as in Section 4.5.2, namely:  $\eta_i = 0.1$ ,  $\eta_f = 0.01$ ,  $\sigma_i = 1/2\sqrt{20^2 + 40^2}$  and  $\sigma_f = 1$ , with training enduring 10 and 40 ordering and tune epochs, respectively.

Figure 4.13 presents the *U-Matrices* of SOM models obtained from the micro-categories generated in the time interval  $t = [0, 20\,000]$  for the *d5k20* data stream, with varying  $q$  values. In the sequence of the discussion in Section 4.5.4, we can confirm that  $q = \{20, 50\}$  allows the detection of 20 clear clusters, while using  $q = 10$  does not allow to derive the expected cluster structure.

Other examples of obtained models and U-matrices for stationary data streams are presented in Figure 4.14, namely for the *Gauss*, *Complex* and *Chain* data streams. Together the results indicate that indeed the presented methodology of micro-categories and weighted batch update rule is able to capture the underlying cluster structure and match the input space density. The later is more obvious in the *Gauss* data stream model with the lattice concentrating more neurons in the denser input space region; in this case the resulting U-Matrix is unintelligible because there is a single cluster in the underlying distribution. U-Matrices for the *Complex* and *Chain* data streams clearly indicate, 7 and 2 clusters, respectively, which conforms with the expected outcome.

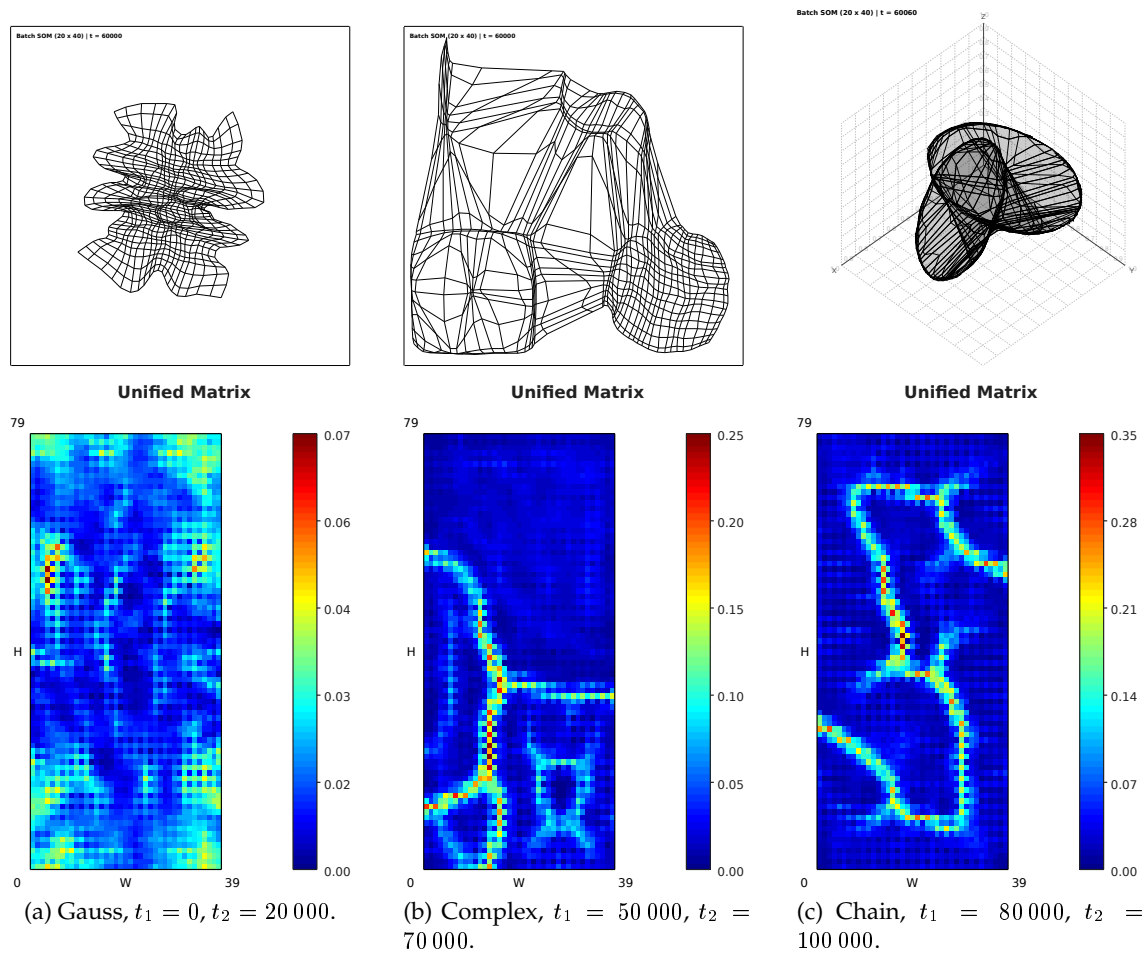


Figure 4.14: Batch SOM models, and derived U-Matrices, generated from specific time intervals over the stationary *Gauss*, *Complex* and *Chain* data streams.

In respect to the non-stationary data streams more than one interval was used to produce the offline SOM models in the attempt to derive the evolution of the underlying cluster structure. Figure 4.15a depicts the U-Matrices obtained from models generated from the set  $t = \{[0, 20\,000], [80\,000, 100\,000], [130\,000, 150\,000]\}$  of time intervals for the *Clouds* data stream. The expected evolution from 2/3/2 clusters can be derived from the visualizations. As for the *Hepta* data stream two time intervals were used, namely before and after the disappearance of the central cluster, e.g.,  $t = \{[80\,000, 100\,000], [100\,000, 120\,000]\}$ , and results are presented in Figure 4.15b. Again, the correct evolution from 7 to 6 clusters is correctly identified. The previous identified cluster structures are correct and can be confirmed by the data streams illustrations and in Appendix B by attending to the extracted intervals.

#### 4.5.6 Model Assessment

The resulting average quantization values  $\overline{qe}(t)$  of the model assessment methodology in Section 4.5.6 are presented in this section. Four data streams were chosen to illustrate the proposed method which are either stationary or contain different types of changes in the underlying cluster structure over time. All results used  $T = 500$  aiming at a medium-term change indication, where present. The codebooks were generated by a StreamART2A algorithm with ground parameters  $L = 1000$ ,  $q = 50$  and  $\eta = 0.05$ . The computation of the  $\overline{qe}(t)$  metric uses the current observation and the corresponding closest micro-category found in the current codebook  $\mathcal{K}$ . The size of the codebook was set with  $K = 1000$ , hence it contains micro-categories for a time horizon of 20 000 past summarized observations. Please note that there are only micro-categories in the codebook after the processing of the first landmark window and the  $\overline{qe}(t)$  metric needs  $T$  observations to produce the initial value. Consequently, the following  $\overline{qe}(t)$  values can only be computed after  $L + T$  observations have been presented to the StreamART2A algorithm.

The metric was proposed with the premise that the quantization quality of the codebook will improve, at least until it reaches its maximum size, if the underlying distribution is stationary, i.e., the error of the “learner” decreases as it learns more observations. If the underlying distribution changes, we expect that current micro-categories in the codebook temporarily (until sufficient new ones are added) fail to properly represent the new observations. Hence, the  $\overline{qe}(t)$  resulting curve should rise as an indication of change.

Figure 4.16a presents the results for the stationary *Complex* data stream. Indeed, the  $\overline{qe}(t)$  values decrease until the codebook is full, i.e.,  $t = 20\,000$ , and the curve is stable throughout the remaining learning procedure. In Figure 4.16b, depicting the result for the *AbruptOneTwo* data stream, we can also observe the same decreasing behavior. However, the abrupt change in the data stream at  $t = 50\,000$  is immediately indicated by the  $\overline{qe}(t)$  curve, followed by a similar decrease in the new stationary phase of the data stream. The *Clouds* data stream contains a gradual change in the underlying distribution from  $t =$

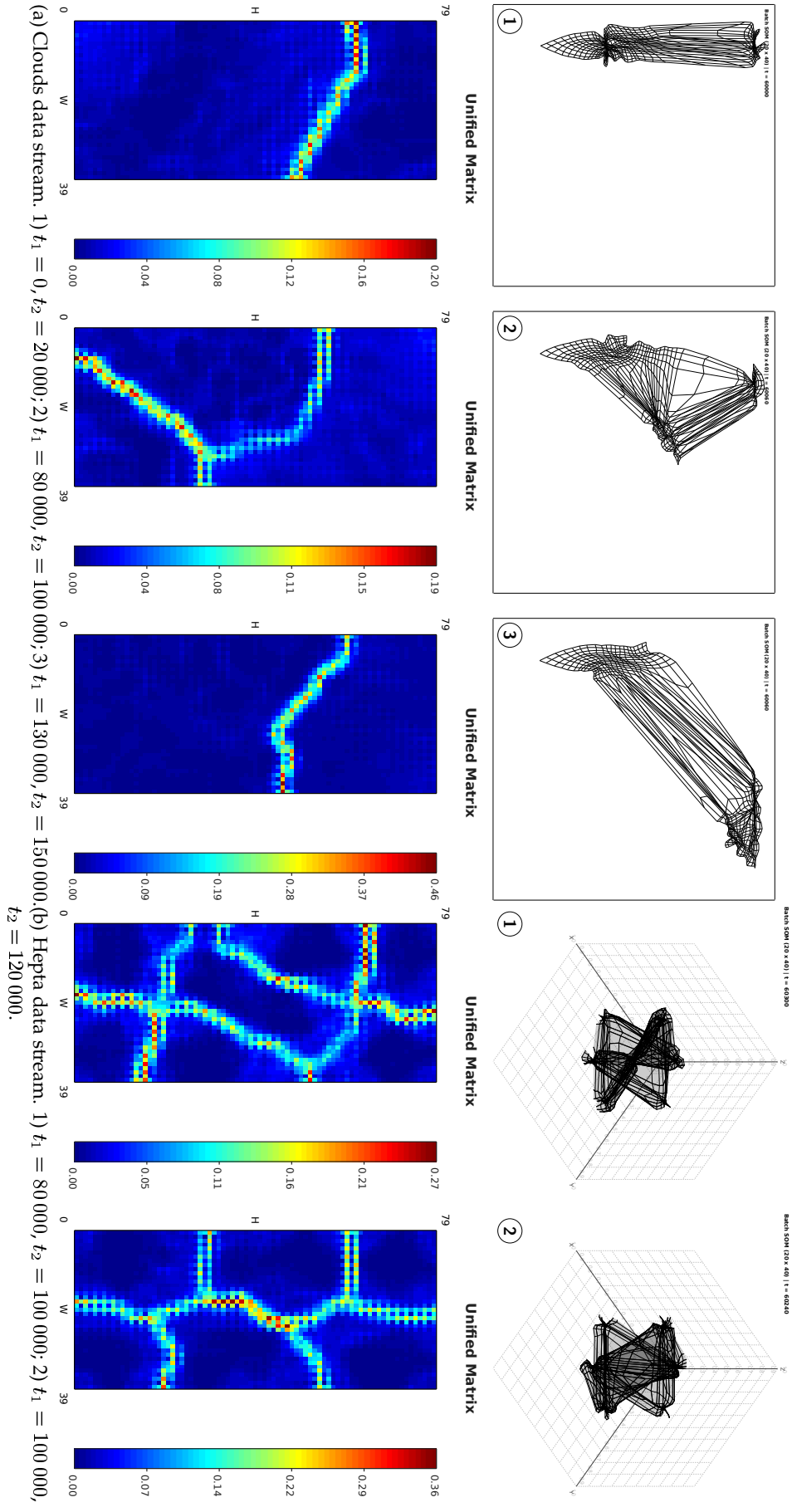


Figure 4.15: U-Matrices derived from Batch SOM models generated from specific time intervals over the non-stationary *Clouds* and *Hepta* data streams.

50 000 to  $t = 150\,000$ , where two of the clusters centers move throughout the input space. This type of gradual change is harder to detect, but the sequence of  $\overline{qe}(t)$  values illustrated in Figure 4.16c actually indicate a gradual and continuous change in that period. Finally, the *Hepta* data stream presents a even harder change to detect with this metric alone. Until  $t = 100\,000$  the observations are drawn from an underlying distribution consisting of 7 *Gaussian* clusters with equal probability. At the change-point one clusters disappears, but the corresponding probability is divided equally among the remaining 6 clusters. Results are illustrated in Figure 4.16d and we can see that the error does not increase, but decreases slightly, because the available micro-categories will now have to represent only 6 clusters. Hence, improvements are necessary in this model assessment methodology in dealing with the disappearance of clusters and an overall automatic change detection mechanism. Both these aspects are discussed in the next section.

## 4.6 Remarks and Future Work

In this chapter the popular two-phase approach to data streams (see Section 2.6) was replicated towards using the SOM to perform offline exploratory cluster analysis. The data stream abstraction procedure is performed by a constrained ART2-A algorithm, called StreamART2A, that introduces the concept of micro-category. A set of  $K$  micro-categories — the codebook or StreamART2A model, is maintained and updated by a continuous procedure that generates representative micro-categories within consecutive landmark windows. This allows the codebook to represent only more recent information and implicitly deal with the non-stationary characteristics of data streams. A comprehensive evaluation of the abstraction procedure performed by the StreamART2A algorithm was performed, including a parameter sensitivity analysis from where recommended parameters were proposed. These parameters should work well across many different problems. Exploratory cluster analysis performed with the SOM over selected periods of the artificial data streams confirmed the expected clustering results.

In this chapter only the clustering of observations task was addressed using artificial data streams. Application of the presented methodology to real-world data can be found in Chapter 7, after the feature clustering methodology for SOM models is presented in Section 5.6.

There are some open issues that are reserved for future work. These are addressed in the following paragraphs.

**Noisy data.** As suggested in Section 4.3.3, using a sufficient high  $q$  value, e.g.,  $q = 50$ , some noisy observations (e.g., common in sensor applications) can be detected within micro-categories, attending on the respective number of observations they represent. By the competitive learning scheme, which minimizes the MSE, it is expected that micro-categories represent in average the same number of observations. However, due to the

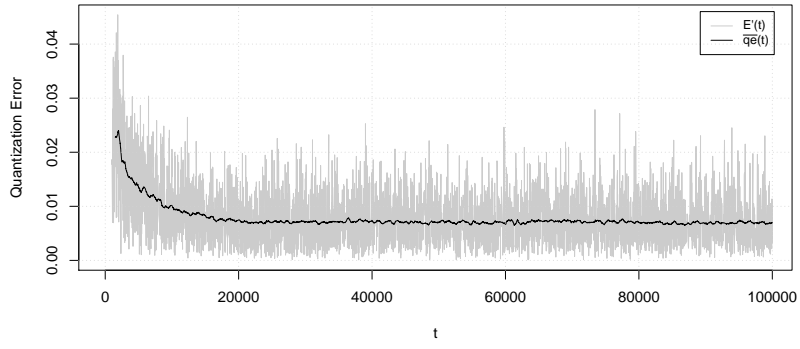
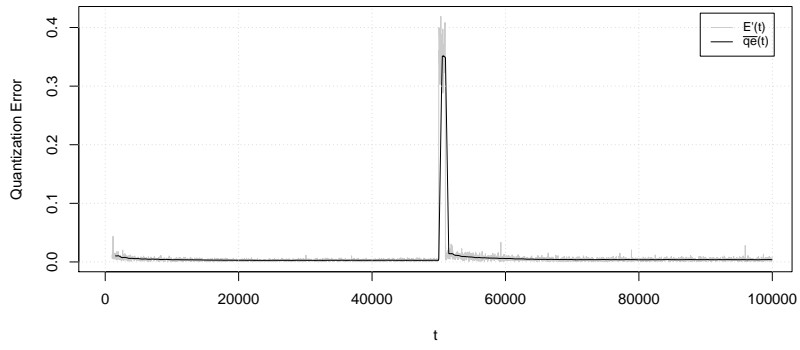
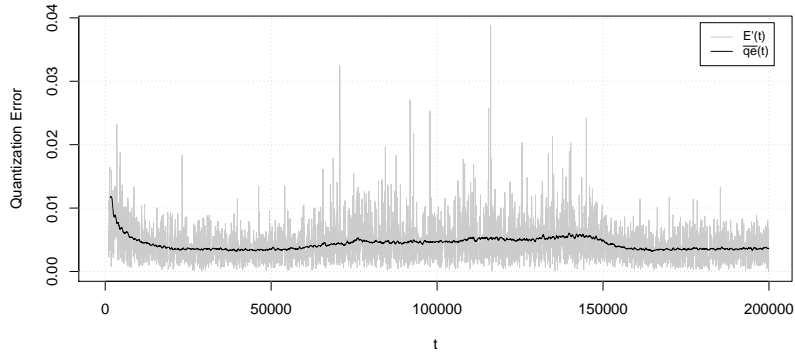
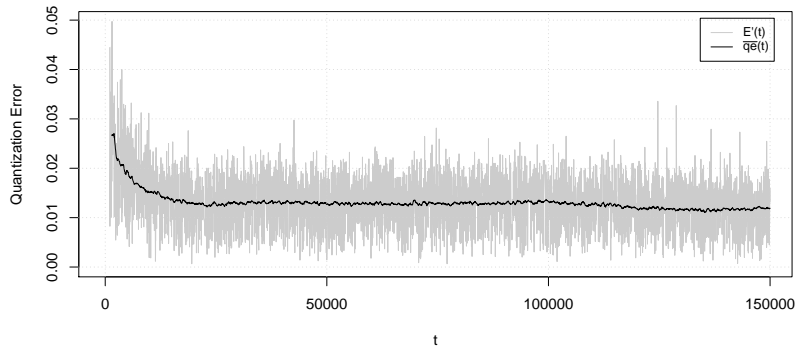
(a) *Complex* data stream.(b) *AbruptOneTwo* data stream.(c) *Clouds* data stream.(d) *Hepta* data stream.

Figure 4.16: Behavior of  $E'_q(t)$  and  $\bar{q}e(t)$  over various data streams with  $L = 1000$ ,  $q = 50$ ,  $\eta = 0.05$ ,  $K = 1000$  and  $T = 500$ .



dynamic creation of micro-categories this may not uphold in practice. Consequently, if a noisy observation (in particular an outlier) arrives in the beginning of a landmark window, there is a high probability that it will be “contained” inside a single micro-category, far apart from the others created. In this case, at the end of the landmark window one can discard micro-categories that represent one or a small number of observations, based on some threshold. This is similar to strategies employed by grid-based methods, e.g., *D-Stream* (see Section 2.6); however, discarding micro-categories can have an impact on the underlying data structure used to hold the codebook — as discussed in Section 4.4.1, regarding the relation between  $q$  and  $K$ . On the other hand, if noisy observations are processed after the  $q$  micro-categories have been reached, then they will have to be incorporated in an existing micro-category or cause merges. In this later case, the vigilance parameter may be altered artificially by the presence of noise, which has a global impact within that landmark window. This problem may be attenuated in subsequent landmark windows, as the *offline* models will necessarily use micro-categories from more than a single landmark window (very few are generated inside each landmark window, not sufficient to generate a large SOM model, as the ones explored). Therefore, the impact of noise on the StreamART2A procedure is a problem that should be studied.

**Model assessment.** From the results presented in Section 4.5.6, it became clear that the model assessment strategy based on  $\overline{qe}(t)$  values may not indicate change in a consistent manner, i.e., increase of the  $\overline{qe}(t)$  values in the presence of change. For example, the disappearance of clusters decreases the error (recall Figure 4.16d).

Some possible strategies for automatic change detection over  $\overline{qe}(t)$  values should always expect error increase in the presence of change; otherwise, the convergence of the model would be seen as change. In the next chapter, when the UbiSOM algorithm is presented, this problem is solved by introducing another assessment metric used in conjunction with the  $\overline{qe}(t)$  metric (namely, in Section 5.3.3). In the case of the StreamART2A model assessment procedure, this additional metric would require that we monitor the ratio of micro-categories (belonging to the codebook) that have been active in the last  $T$  observations; “active” meaning a micro-category was selected as the BMU<sup>3</sup> in the  $\overline{qe}(t)$  computation. If a high number of micro-categories cease to be active, then this implies that some region in the input space has disappeared. As such, this technique would allow the previous required consistency in error increase in the presence of any type of change, and may be explored in future work.

**Automatic change detection.** With automatic change detection mechanisms, upon detection, a snapshot of the codebook  $\mathcal{K}$  can be stored for later *offline* analysis. Otherwise, the time horizon from which *offline* models can be generated is limited by the StreamART2A parameterization, namely by Eq. (4.9). Towards this improvement, some suggestions are made regarding developing or using existing separate methods in future

<sup>3</sup>This SOM terminology was used to convey the idea in a more succinct way.

work.

Some time was devoted pursuing automatic change mechanisms that could be applied to the *average quantization error*  $\overline{qe}(t)$ . In (Silva, Marques, and Panosso 2012) the StreamART2A algorithm was employed over a data stream containing the *Dow Jones Index* values and other computed technical statistics, with the overall goal of detecting change in financial markets. The methodology consisted in computing two average quantization errors  $\{\overline{qe}_1, \overline{qe}_2\}$  of different lengths, i.e.,  $T_2 \gg T_1$ , so as to obtain a short and a relatively longer trend. By subtracting the latter from the former, a detection threshold around the value of zero is obtained, i.e., when the subtraction exceeds zero, change is detected. This methodology stems from a widely used technique in financial markets to detect changes in the trend of stock values. While this application was successful in detecting changes, the estimation of appropriate window sizes is not straightforward and may require expert domain knowledge.

Other strategies were explored, namely some inspired by statistical process control (Bošnjak and Cisar 2010). Let an anomalous behavior be signaled by:

$$E'_q(t) \geq (\alpha + 1)\overline{qe}(t)$$

where  $0 < \alpha < 1$  is the percentage above the average quantization error that is considered anomalous. This, by itself, could lead to false positive alarms in the presence of noise. Hence, change can be signaled when anomalous behavior is detected for  $C$  consecutive violations of the threshold:

$$\sum 1_{n=t-C+1}^t \{E'_q(n) \geq (\alpha+1)\overline{qe}(n)\} \geq C.$$

Such strategy makes it very easy to automatically detect abrupt changes in the data stream, e.g., Figure 4.16b. However, gradual changes, e.g., Figure 4.16c, are harder to detect with this strategy. If the previous anomalous behavior is instead defined by  $\overline{qe}(t) \geq (\alpha + 1)\overline{qe}_{min}$ , then gradual changes may be possible to detect.

Notwithstanding these possible directions, there are other generic proposals in literature that can potentially be used, such as the use of *martingales* (Ho and Wechsler 2007) or differences between estimated distributions inside two consecutive sliding windows (Kifer, Ben-David, and Gehrke 2004); differences are based on statistical tests, e.g., *Kolmogorov-Smirnov* test. However, they are completely separate procedures with their own time and space complexity costs that do not make use of assessment metrics that could be derived from the StreamART2A codebook. All these methods (discussed and cited) are non-parametric. Parametric change detection in unsupervised learning may be impossible, e.g., the errors  $E'_q(t)$  do not fit into any known distribution — this was verified experimentally during the research.



# The Ubiquitous Self-Organizing Map for Non-Stationary Data Streams

A belief is not merely an idea the mind possesses; it is an idea that possesses the mind.

---

ROBERT OXTON BOLT, ENGLISH PLAYWRIGHT (1924-1995)

The previous methodology already allows SOM models to be obtained in a stream setting, although in an offline manner. This imposes a delay in obtaining such models for exploratory cluster analysis. Hence, a SOM variant that can learn directly for data streams is of great interest for, e.g., real-time monitoring applications. However, the classical Online SOM algorithm relies on time-dependent annealing schemes to guide the evolution of learning parameters, so as to ensure convergence of the map. Consequently, it implicitly assumes the underlying distribution is stationary. This chapter addresses the problem of learning non-stationary data streams with the SOM, towards the established requirements for such variants.

## 5.1 Chapter Overview

This chapter presents and evaluates the *Ubiquitous Self-Organizing Map* (UbiSOM) ([Silva and Marques 2015a](#); [Silva and Marques 2015b](#)), a novel SOM algorithm tailored for non-stationary multidimensional data streams.

The chapter is organized as follows. In the next section, motivation and aims are

presented, which support the methodology. In Section 5.3, the proposed model and algorithm are detailed and formalized, after which some considerations are made in Section 5.4 regarding the dynamics of the algorithm and its computational complexity. A comprehensive validation and experimental evaluation with artificial data streams is presented in Section 5.5, targeting parameter sensitivity analysis, convergence over stationary and non-stationary data streams, visual exploratory cluster analysis and comparison with existing variants. Section 5.6 addresses the clustering of features (related to time series clustering), namely the methodology and evaluation with artificial data. In Section 5.7, after the algorithm is presented and evaluated, its characteristics are compared against the proposed requirements of Section 2.5.1. Finally, Section 5.8 concludes the chapter, including foreseen future work.

## 5.2 Motivation and Aim

In Chapter 4 the use of the SOM for exploratory cluster analysis was addressed in a two-phase approach to data streams, where SOM models are obtained offline from summarizations provided by the StreamART2A algorithm. This is a viable approach comparable with other surveyed methods (Section 2.6), but leveraging the powerful data visualization abilities of the SOM. However, some applications may benefit from a real-time use of the SOM in streaming environments, e.g., monitoring applications, where cluster analysis can be performed readily from the SOM visualizations. This is supported by the fact that the SOM codebook may be regarded as one of the best abstraction models, due to the topology-preserving mapping and readily derived visualizations, as discussed in Section 2.7. Also, the feature clustering capabilities of the SOM can be explored in this setting, which are intrinsically related to clustering time series (see Section 2.6.2). Hence, a SOM variant that learns directly from non-stationary data streams is of significant relevance.

In Section 2.5.1, desirable requirements SOM variants should possess for this purpose were proposed. The original SOM algorithm uses annealing schemes for the learning parameters  $\eta(t)$  and  $\sigma(t)$  to ensure convergence. Although this results in a stable learning process, fixed annealing schemes require that the total number of observations is known, which is not the case with unbounded data streams. Even if an annealing scheme is used only in the first iterations and then learning parameters remain with low values in subsequent iterations, this completely removes the SOM ability to react to change in the underlying distribution, because the magnitude of the prototype updates, i.e., loses *plasticity*. Other variants, surveyed in Section 2.5.2, that can deal with non-stationarity and use time-independent learning parameter estimation, either alter the SOM lattice regular structure, by dynamically adding or removing neurons — which hinders the standard visualization capabilities, or maintain the regular structure and estimate the parameters based on the local quantization error  $E_q(t)$  alone, namely the PLSOM (Berglund 2010)

and DSOM (Rougier and Boniface 2011). In theory, the later possess an indefinite plasticity to cope with change, but by using only the local error they do not ensure the density matching property of the original SOM, which can also affect the quality of the visualizations for exploratory cluster analysis.

Consequently, in Section 2.7.1, another path of research was established in order to devise a new variant of the SOM algorithm that could overcome the above limitations, which is the main objective of this chapter. The main idea expressed was to use global assessment metrics from where learning parameters are monotonically increased or decreased dependent on the adaptation of the SOM lattice to the underlying distribution. One of these metrics, namely the *average quantization error*  $\overline{qe}(t)$ , was introduced in Section 4.3.5 and the results showed that its behavior matches the above desired behavior that we wish to translate to the learning parameters. However, it does not allow to detect all types of change, e.g., disappearance of clusters. Global assessment metrics can express the fitness of the entire lattice to the underlying distribution, instead of local assessments used by PLSOM and DSOM; also, they can potentially allow the monotonic decrease of learning parameters during stationary phases of the data stream in order to ensure density matching. Consequently, the main goal of this chapter is dependent of the fulfillment of another goals, namely: devising global metrics to assess the fitness of the model and establishing how these metrics can be balanced to estimate learning parameters along time, ensuring the model can react to the non-stationary aspect of data streams.

## 5.3 The Ubiquitous Self-Organizing Map

This chapter addresses the problem of maintaining a SOM model over potentially unbounded non-stationary data streams, by continuously estimating its learning parameters adequately. The main intuition supporting this research is that the “fitness” of the model to the underlying distribution should be continuously monitored and learning parameters adjusted accordingly, either to allow for convergence in stationary phases or to increase plasticity in non-stationary phases.

### 5.3.1 Algorithm Overview

The proposed UbiSOM algorithm relies on two global assessment metrics, namely the *average quantization error* and the *average neuron utility*, computed over a sliding window. While the first assesses the trend of the vector quantization process towards the underlying distribution, the later is able to detect whether regions of the map have become “unused”, given some changes in the distribution not detected by the previous metric, e.g., disappearance of clusters. Both metrics are weighed by a *drift function* that gives an overall indication of the performance of the map over the data stream, which is then effectively used to estimate learning parameters.

Moreover, the UbiSOM implements a finite state-machine consisting of two states,

namely *ordering* and *learning* states. The *ordering* state allows the map to initially unfold over the underlying distribution with monotonically decreasing learning parameters; it is also used to obtain the first values of the assessment metrics, transitioning afterwards to the *learning* state. Hereafter, the learning parameters are decreased or increased based on the *drift function*. This allows the UbiSOM to retain an indefinite *plasticity*, while maintaining the original SOM properties, over non-stationary data streams. These states also coincide with the two typical training phases suggested by *Kohonen* (see Section 2.4.4). It is possible, however, that unrecoverable situations from abrupt changes in the underlying distribution are detected, which leads the algorithm to transition back to the *ordering* state.

### 5.3.2 Notation

The UbiSOM operates over a multidimensional stream of continuous real-valued set of observations  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ , potentially unbounded ( $N \rightarrow \infty$ ), from a manifold  $\Omega \in \mathbb{R}^d$ . Each observation is described by a  $d$ -dimensional feature vector  $\mathbf{x}(t) = [x_t^j]_{j=1}^d$ . As with the classical model, UbiSOM establishes a topological ordered projection from the input space manifold onto a set  $\mathcal{K}$  of  $K$  “extended” neurons (the codebook), arranged in a *width*  $\times$  *height* regular lattice. Each neuron  $k$  in the UbiSOM model is a tuple  $\mathcal{W}_k = \langle \mathbf{w}_k, t_k^{update}, t_k^{bmu} \rangle$ , where  $\mathbf{w}_k \in \mathbb{R}^d$  is the data prototype,  $t_k^{update}$  stores the *timestamp* of the last time its prototype was updated and  $t_k^{bmu}$  stores the last time it was selected as the BMU. The latter is not required in the methodology presented in this chapter, only in collaborative and distributed learning mechanisms presented in Chapter 6, namely to filter unrepresentative prototypes.

We assume all features of the data stream are equally normalized between  $[x'_{min}, x'_{max}]$ , as discussed in Section 3.5. For each incoming observation  $\mathbf{x}(t)$ , presented at time  $t$ , two metrics are computed, within a sliding window of length  $T$ , namely the *average quantization error*  $\overline{qe}(t)$  and the *average neuron utility*  $\overline{\lambda}(t)$ . The former was already introduced in Section 4.3.5 and is adapted for the UbiSOM; the later  $\overline{\lambda}(t)$  metric averages neuron utility  $\lambda(t)$  values that are computed as a ratio of updated neurons during the last  $T$  observations. Both metrics are defined between  $[0, 1]$  and are used in the *drift function*  $d(t)$ , where the parameter  $\beta \in [0, 1]$  weighs the contribution of each metric.

The UbiSOM switches between the *ordering* and *learning* states, both using the classical *Online SOM* update rule, but with different mechanisms for estimating learning parameters  $\sigma(t)$  and  $\eta(t)$ . The *ordering* state endures for  $T$  observations, until the first values of  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$  are available, establishing an interval  $[t_i, t_f]$ , during which monotonically decreasing functions  $\sigma(t)$  and  $\eta(t)$  are used to decrease values between  $\{\sigma_i, \sigma_f\}$  and  $\{\eta_i, \eta_f\}$ , respectively. The *learning* state estimates learning parameters as functions of the *drift function*. Also, the UbiSOM neighborhood function is defined in such a way that  $\sigma(t) \in [0, 1]$ , as opposed to all existing variants, where the domain of these values is dependent of the lattice size. The notation is summarized in Table 5.1.

Notation	Description
$d$	data stream <i>dimensionality</i>
$t$	<i>time</i> , i.e., iteration number
$\mathbf{x}(t)$	<i>observation</i> presented at time $t$ , where $\mathbf{x}(t) \in \mathbb{R}$
$width$	<i>width</i> of the lattice
$height$	<i>height</i> of the lattice
$K$	total number of neurons, i.e., $width \times height$
$k$	neuron <i>index</i>
$c$	index of <i>BMU</i>
$\mathcal{W}_k$	UbiSOM “extended” neuron: a tuple $\langle \mathbf{w}_k, t_k^{update}, t_k^{bmU} \rangle$
$\mathbf{w}_k$	prototype of neuron $k$
$t_k^{update}$	<i>time stamp</i> of last prototype update
$t_k^{bmU}$	<i>time stamp</i> of last selection as BMU
$E'_q(t)$	<i>normalized local quantization error</i> for $\mathbf{x}(t)$
$ \Omega $	input manifold diagonal, the normalization value used by $E'_q(t)$
$\overline{qe}(t)$	<i>average quantization error</i> at time $t$
$\lambda(t)$	<i>neuron utility</i> at time $t$
$\overline{\lambda}(t)$	<i>average neuron utility</i> at time $t$
$d(t)$	<i>drift function</i> at time $t$
$t_i$	<i>initial time</i> of the ordering state
$t_f$	<i>final time</i> of the ordering state
<b>Algorithm specific parameters</b>	
$T$	size of sliding window / length of ordering state
$\beta$	drift function <i>weight factor</i>
$\sigma_i$	<i>initial neighborhood</i> for ordering state
$\sigma_f$	<i>final neighborhood</i> for ordering state
$\eta_i$	<i>initial learning rate</i> for ordering state
$\eta_f$	<i>final learning rate</i> for ordering state

Table 5.1: Notation for the UbiSOM algorithm.



### 5.3.3 Online Assessment Metrics

The purpose of the following metrics is to assess the “fitness” of the current UbiSOM model to the underlying distribution. Both proposed metrics are computed over a sliding window of length  $T$  and are later used to estimate learning parameters in the UbiSOM algorithm.

#### Average Quantization Error

The *average quantization error* metric gives an indication of how well the map is currently quantifying the underlying distribution. In most situations where the underlying data stream is stationary, the metric is expected to decrease and stabilize. On the other hand, if the shape of the distribution changes, it is expected to increase.

The widely used mean quantization error ( $\overline{QE}$ ) metric (see Section 2.4.6) is the standard measure of fit of a SOM model to a particular distribution. It is typically used to compare SOM models obtained for different runs and/or parameterizations and used in a *static* data setting. The rationale is that the model which exhibits a lower  $\overline{QE}$  value is better at abstracting the input space.

Regarding data streams, this metric, as it stands, is not applicable because data is potentially infinite. Competing approaches to the proposed UbiSOM, name PLSOM and DSOM — Section 2.5.2, estimate learning parameters proportionally to the local quantization error  $E_q(t)$ . However, the local error is very unstable because the topological projection and quantization is a many-to-few mapping, where some observations are better represented than others. Moreover, considering stationary distributions, Kohonen recommended that both  $\eta(t)$  and  $\sigma(t)$  should decrease monotonically with time, a critical condition to achieve convergence (Kohonen 2001). As an example, with stationary data the local error does not decrease monotonically over time, just the overall variance, e.g., see Figure 5.1 until  $t = 50\,000$ .

In the UbiSOM algorithm, the mean quantization error was modified to a moving average in the form of the proposed *average quantization error*  $\overline{qe}(t)$ , based on the premise that the error of a learner will decrease over time for an increasing number of examples if the underlying distribution is stationary; otherwise, if the distribution changes, the error increases. For each observation  $\mathbf{x}(t)$  the  $E'_q(t)$  local quantization error is obtained during the BMU search, as the normalized Euclidean distance:

$$E'_q(t) = \frac{\|\mathbf{x}_t - \mathbf{w}_c\|}{|\Omega|} \quad (5.1)$$

where  $|\Omega|$  is the largest diagonal of the normalized input space, as discussed in Section 3.5. These values are averaged over a window of length  $T \gg 1$  to obtain  $\overline{qe}(t)$ , defined in Eq. (5.1). Consequently, the value of  $T$  establishes a short, medium or long-term trend of



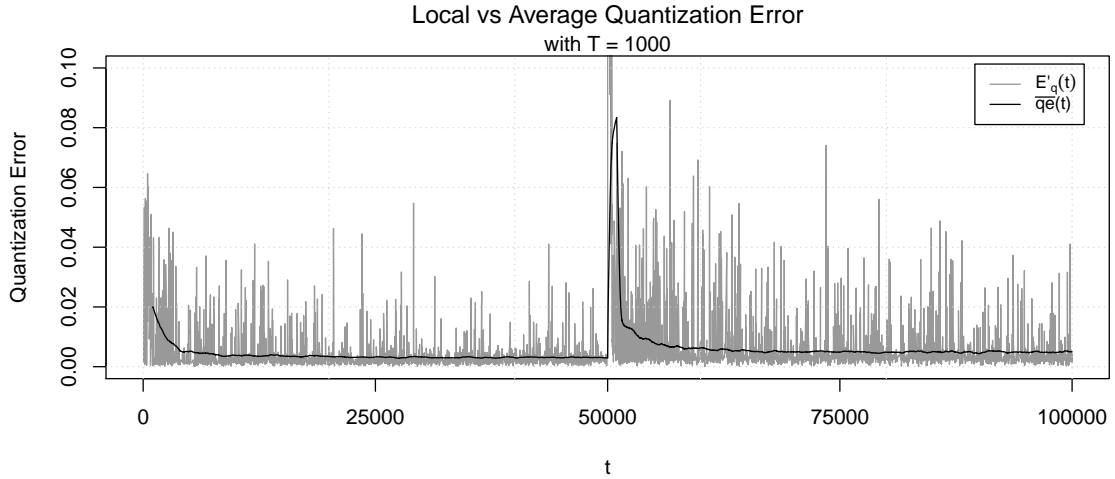


Figure 5.1: Example of behavior of local  $E'_q(t)$  vs. average quantization error  $\overline{qe}(t)$ .

the model adaptation.

$$\overline{qe}(t) = \frac{1}{T} \sum_t^{t-T+1} E'_q(t) \quad (5.2)$$

Despite the moving average also changing during gradual changes of the data streams (as later illustrated in Section 5.5), it shall be exemplified first on an extreme case. Figure 5.1 depicts the behavior of both  $E'_q(t)$  and  $\overline{qe}(t)$  values (for  $T = 1000$ ) during an example run of the UbiSOM algorithm over a data stream containing an abrupt change at  $t = 50\,000$ , between two stationary phases. We can observe that  $E'_q(t)$  values exhibit a large variance throughout time, even during stationary intervals, as opposed to  $\overline{qe}(t)$  which is smoother and indicates the trend of the convergence. Therefore, the proposed metric assumes that if  $\overline{qe}(t)$  is decreasing, then the underlying distribution is stationary; otherwise, it is changing. The first version of the UbiSOM algorithm proposed in (Silva and Marques 2015a) only used this metric. However, it may fail to detect all types of changes, therefore leading to the proposal of the following additional assessment metric.

### Average Neuron Utility

The *average neuron utility* is an additional measure that gives an indication of the proportion of neurons that are actively being updated. The decrease of this metric indicates neurons with old information, which can reflect changes in the underlying distribution not detected by  $\overline{qe}(t)$ .

The  $\overline{qe}(t)$  metric may be a good overall indicator of the fitness of the model. Despite that, it may be unable to detect the abrupt disappearance of clusters. For example, if we have a stationary distribution composed of, e.g., seven *Gaussian* clouds of data points and a UbiSOM model has converged over it, the  $\overline{qe}(t)$  should be stable. If one of the clouds disappears, there will be no increase in the  $\overline{qe}(t)$  trend, since the “assigned” neurons for

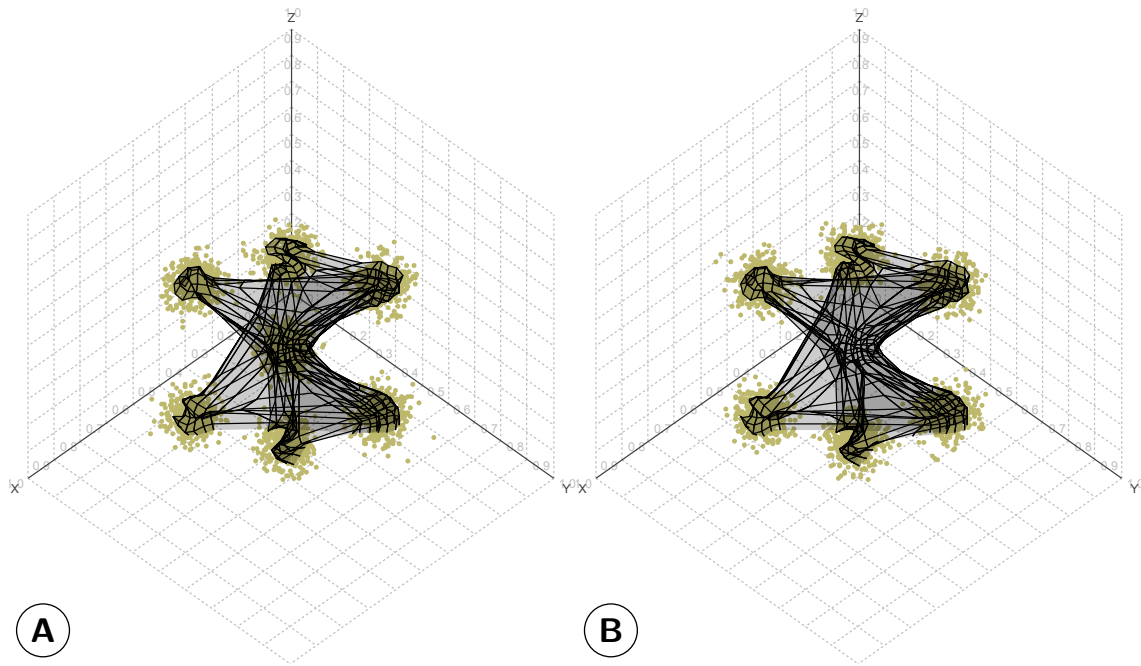


Figure 5.2: Example of a distribution change not detected by the *average quantization error*. (A) Before change; (B) After the disappearance of the inner cluster, where it is visible a region of unused neurons.

each cluster remain the same. Figure 5.2 illustrates this scenario, depicting an area of the map with neurons that do not represent current observations, after the inner cluster disappears. In this situation, the learning parameters should increase and allow the map to recover from this situation. The *average neuron utility*  $\bar{\lambda}(t)$  was introduced to address these situations.

To compute this assessment metric each UbiSOM neuron  $k$  contains a *timestamp*  $t_k^{update}$  which stores the last time the corresponding prototype was updated, functioning as an *aging* mechanism — this type of mechanism is popular in other SOM variants that add and remove neurons dynamically (see Section 2.5.2). A prototype is updated if it is the BMU or if it falls in the influence region of the BMU, limited by the neighborhood function. Initially,  $t_k^{update} = 0$ .

The neuron utility  $\lambda(t)$  is computed with Eq. (5.3). It measures the ratio of neurons that were updated within the last  $T$  observations, over the total number of neurons. Consequently, if all neurons have been recently updated, then  $\lambda(t) = 1$ . The values are then averaged to obtain  $\bar{\lambda}(t)$ , as formalized in Eq. (5.4).

$$\lambda(t) = \frac{\sum_{k=1}^K 1_{\{t - t_k^{update} \leq T\}}}{K} \quad (5.3)$$

$$\bar{\lambda}(t) = \frac{1}{T} \sum_t^{t-T+1} \lambda(t) \quad (5.4)$$

As a result, a decrease in  $\bar{\lambda}(t)$  indicates that there are neurons not being used to quantize the data stream. In literature, these neurons are often called “dead-units” and the existence of some of these neurons (units) may be considered normal when the SOM converges over a relatively complex distribution, i.e., some units may lay between clusters and never chosen as a BMU after convergence (recall example of Figure 2.5). Nonetheless, a decreasing trend should alert for changes in the underlying distribution.

### On the Use of Gaussian Moving Averages

Both  $\bar{q}e(t)$  and  $\bar{\lambda}(t)$  effectively act as *linear low-pass filters* over a set of continuous values. However, the presence of higher  $E'_q(t)$  values, either caused by noise or abrupt change, can also lead to non-differentiable responses of these metrics. With the goal of monotonically decreasing/increasing learning parameters, we wish to avoid this behavior. Smoother filter responses can be obtained with the use of *Gaussian filters*, at the expense of a higher computational cost. A Gaussian filter is considered the ideal time domain filter whose impulse response is an approximation of a Gaussian function (Blinchikoff and Zverev 2001). It also has additional advantages when compared to linear filters, namely minimizing the rise and fall time of the response. To overcome the additional computation overhead, the fact that Gaussian filters can be approximated by cascading linear filters, with decreasing window lengths by a specific factor (Wells 1986), was explored.

Consequently, instead of using one linear filter, i.e., simple moving average, of length  $T$ , we can use a triple cascaded moving average of the same length to obtain a *Gaussian* filter. Figure 5.3 depicts the same example of previous Figure 5.1, zoomed at the change point, and two implementations for  $\bar{q}e(t)$ : a simple moving average (MA) and a triple cascaded moving average (3MA), both with  $T = 1000$ . The later indeed exhibits better properties to estimate learning parameters, namely a smoother response, e.g., differentiable at all points, and minimization of rise and fall time in the presence of true change. Also, sporadic high  $E'_q(t)$  values caused by noise in the data stream, are also further smoothed out by the 3MA. Appendix D contains additional information and implementation details on this particular moving average.

Thus, this research proposes the use of a 3MA for the assessment metrics  $\bar{q}e(t)$  and  $\bar{\lambda}(t)$  and all subsequent results for the UbiSOM algorithm also rely on a 3MA implementation.

#### 5.3.4 The Drift Function

The previous metrics  $\bar{q}e(t)$  and  $\bar{\lambda}(t)$  are both weighed in a *drift function* that is used by the UbiSOM to estimate learning parameters. The *drift function* is defined as:

$$d(t) = \beta \bar{q}e(t) + (1 - \beta) (1 - \bar{\lambda}(t)) \quad (5.5)$$

where  $\beta \in [0, 1]$  is a parameter that establishes the balance of importance between the two metrics. Since both  $\bar{q}e(t)$  and  $\bar{\lambda}(t)$  are only obtained after  $T$  observations, so is  $d(t)$ .

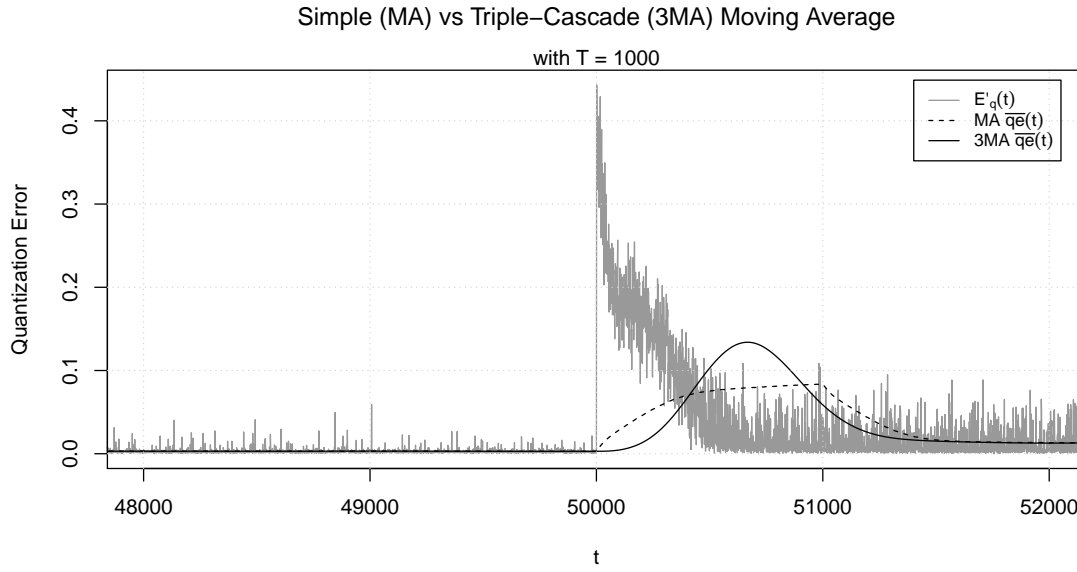


Figure 5.3: Behavior of a simple moving average *versus* a triple cascaded moving average in the presence of abrupt change.

A quick analysis of  $d(t)$  should be made. First of all, it should be clear that if  $\beta = 1$ , then the drift function is only defined by the  $\bar{q}e(t)$  metric. With high learning parameters, specially the neighborhood  $\sigma(t)$  value,  $\bar{\lambda}(t)$  is expected to be  $\approx 1$ , which practically eliminates the second term of the equation. Consequently, the drift function is only governed by  $\bar{q}e(t)$ . When the neuron utility decreases, the second term contributes to the increase of  $d(t)$  in proportion to the chosen  $\beta$  value. Empirically,  $\beta$  should be parameterized with relatively high values, establishing  $\bar{q}e(t)$  as the main measure of “fitness” and using  $\bar{\lambda}(t)$  as a fail-safe mechanism.

Until the first  $T$  observations have been processed, the UbiSOM does not rely on the  $d(t)$  function to estimate learning parameters, but on monotonically decreasing learning parameters, as presented in Section 5.3.6.

### 5.3.5 The Neighborhood Function

The UbiSOM algorithm uses a normalized neighborhood radius  $\sigma(t)$  learning parameter and a truncated neighborhood function. The latter is what effectively allows  $\bar{\lambda}(t)$  to be computed.

The performed normalization is based on the maximum distance between any two neurons in the lattice. In rectangular maps the farthest neurons are the ones at opposing diagonals, e.g., positions  $(0, 0)$  and  $(width - 1, height - 1)$  in Figure 5.4. Hence distances within the lattice are normalized by the Euclidean norm of the vector  $\mathbf{DIAG} = (width - 1, height - 1)$ , defined as:

$$\|\mathbf{DIAG}\| = \sqrt{(width - 1)^2 + (height - 1)^2}. \quad (5.6)$$

This effectively limits the maximum neighborhood width the UbiSOM can use and establishes  $\sigma \in [0, 1]$ .

The neighborhood function of the UbiSOM variant is given by:

$$h'_{ck}(t) = e^{-\left(\frac{\|r_c - r_k\|}{\sigma(t) \|\text{DIA}\|}\right)^2}. \quad (5.7)$$

Compared to the *Original* SOM neighborhood function (see Eq. 2.5), only the normalization term is introduced. To get a grasp on how different  $\sigma$  values determine the influence region around the BMU, Figure 5.5 depicts Eq. (5.7) for different  $\sigma$  values. Neurons whose values of  $h'_{ck}(t)$  are below a *threshold* of 0.01 are not updated. This is critical for the computation of  $\lambda(t)$ , since  $h'_{ck}(t)$  never reaches zero and all prototypes would still be updated with very small values. The truncated neighborhood function is also a performance improvement, avoiding negligible updates to prototypes. The threshold value is empirical and has no significant consequence in the convergence of the map. For example, the SOM is able to properly converge using the *bubble* function as the neighborhood kernel (Kohonen 2013).

### 5.3.6 States and Algorithm Formalization

The UbiSOM algorithm implements a finite state-machine, i.e., it can switch between two states. This design was, on one hand, imposed by the initial delay in obtaining values for the assessment metrics and, consequently, for the drift function  $d(t)$ . It was also seen as a desirable mechanism to conform to *Kohonen's* proposal of an ordering and a convergence phase for the SOM (see Section 2.4.4). It was furthermore explored to deal with drastic changes that may occur in the underlying distribution.

The UbiSOM algorithm switches between two states, namely *ordering* and *learning*, where the same update rule is used, but with different parameter estimation methods. The update equation is formalized in Eq. (5.8) and uses the neighborhood kernel of Eq. (5.7). Please note that the prototypes are only updated above the established update threshold. It is identical to the *Online* SOM algorithm (see Eq. 2.4), apart from the update threshold.

$$\mathbf{w}_k(t+1) = \begin{cases} \mathbf{w}_k(t) + \eta(t)h'_{ck}(t) [\mathbf{x}_t - \mathbf{w}_k(t)] & \text{if } h'_{ck}(t) > 0.01 \\ \mathbf{w}_k(t) & \text{otherwise} \end{cases} \quad (5.8)$$

Figure 5.6 summarizes and depicts the two possible states of the UbiSOM algorithm and when transitions occur. Each state is described next, together with the respective learning parameter estimation methods.

#### Ordering State

The *ordering state* is the initial state of the UbiSOM algorithm and to where it reverts if it cannot recover from an abrupt change in the data stream. It endures for  $T$  observations,

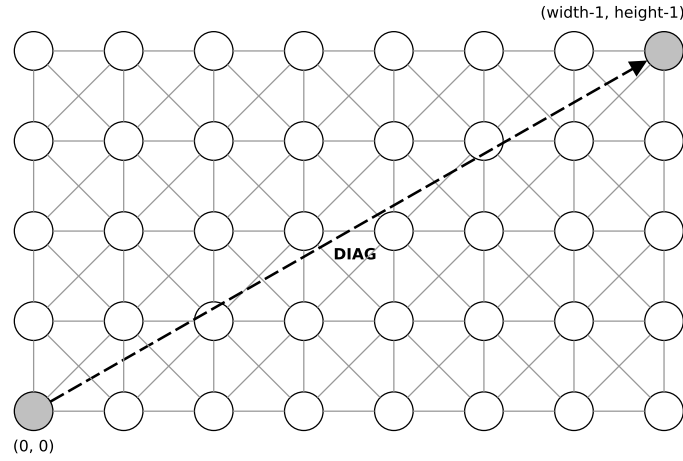


Figure 5.4: Maximum lattice distance  $\| \text{DIAG} \|$  used for normalization of  $\sigma(t)$ .

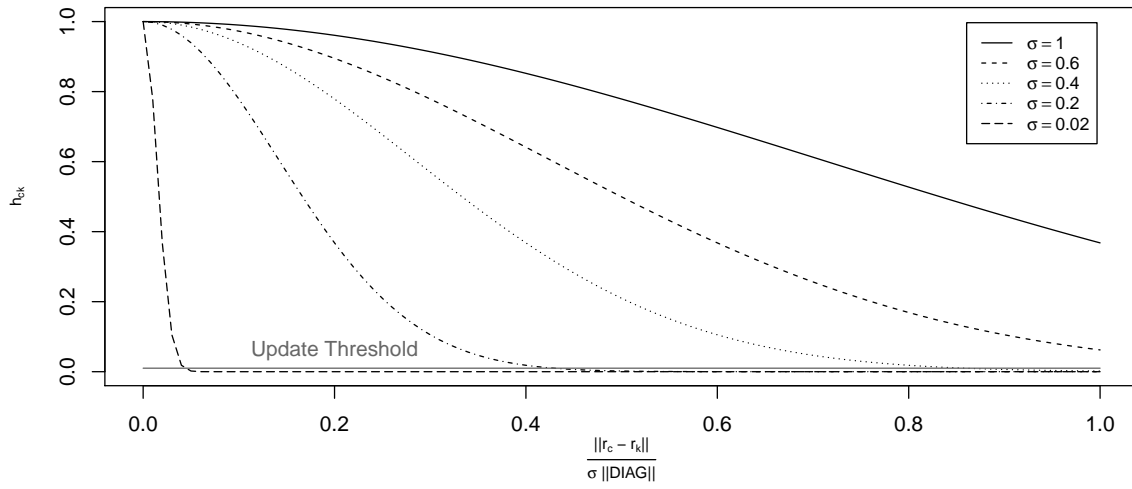


Figure 5.5: UbiSOM neighborhood kernel  $h'_{ck}$  for different values of  $\sigma$ . The update threshold value is also depicted.

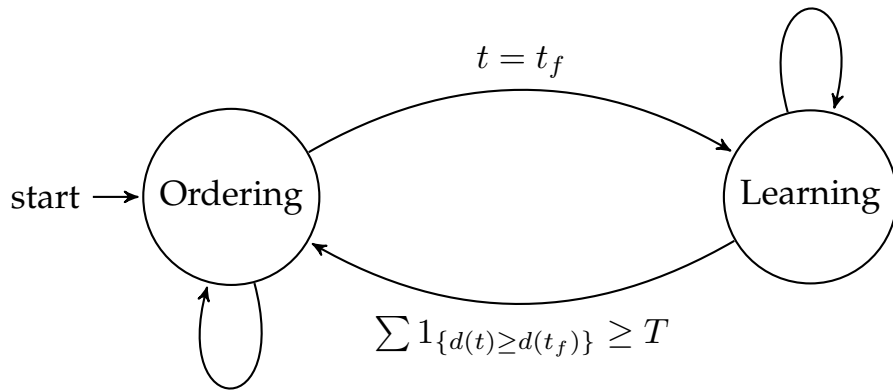


Figure 5.6: UbiSOM states and transitions.

where learning parameters are estimated with a monotonically decreasing function, i.e., time-dependent, similar to the *Online* SOM algorithm. Thus, the parameter  $T$  simultaneously defines the window length of the assessment metrics, as well as dictates the duration of the *ordering state*. The parameters should be relatively high, so the map can unfold from a totally unordered initialization of the prototypes. This phase also allows for the first value of the drift function  $d(t)$  to be available. After  $T$  observations the algorithm switches to the *learning state*.

Let  $t_i$  and  $t_f = t_i + T - 1$  be the first and last iterations of the ordering phase, respectively. This state requires choosing appropriate parameter values for  $\eta_i$ ,  $\eta_f$ ,  $\sigma_i$  and  $\sigma_f$ , which are the initial and final values for the learning rate and the normalized neighborhood radius, respectively. The choice of values will greatly impact the initial ordering of the prototypes and will affect the range of learning parameter values in the *learning state*. Any monotonically decreasing function can be used, but the proposed algorithm uses the decreasing exponential function for both learning parameters:

$$\sigma(t) = \sigma_i \left( \frac{\sigma_f}{\sigma_i} \right)^{t/t_f}, \quad \eta(t) = \eta_i \left( \frac{\eta_f}{\eta_i} \right)^{t/t_f} \quad \forall t \in \{t_i, t_{i+1}, \dots, t_f\} \quad (5.9)$$

The same functions were suggested for the classical SOM algorithms in Section 2.4.4, namely Eq. (2.8), and are effectively used in their implementations.

At the end of the  $t_f$  iteration, the first value of the drift function is obtained and the UbiSOM algorithm transitions to the *learning state*.

### Learning State

The *learning state* begins at  $t_f + 1$  and is the main state of the UbiSOM algorithm, during which learning parameters are estimated in a time-independent manner. Here, learning parameters are estimated solely based on the drift function  $d(t)$ , decreasing or increasing relative to the first computed value  $d(t_f)$  and final values  $(\eta_f, \sigma_f)$  of the *ordering state*.

Learning parameters  $\eta(t)$  and  $\sigma(t)$  are estimated for an observation presented at time  $t$  by Eq. (5.10), where  $d(t)$  was previously formalized in Eq. (5.5). One can easily see that learning parameters are estimated proportionally to  $d(t)$ . Also, final values of the ordering state for  $\eta_f$  and  $\sigma_f$  establish an upper bounded for the learning parameters in this state.

$$\eta(t) = \begin{cases} \frac{\eta_f}{d(t_f)} d(t) & d(t) < d(t_f) \\ \eta_f & \text{otherwise} \end{cases} \quad \sigma(t) = \begin{cases} \frac{\sigma_f}{d(t_f)} d(t) & d(t) < d(t_f) \\ \sigma_f & \text{otherwise} \end{cases} \quad (5.10)$$

The outcome of these equations is that if the distribution is stationary the learning parameters accompany the decrease of the drift function values, allowing the map to converge to a stable state. On the contrary, if changes occur, the drift function values rise, consequently increasing the learning parameters, and increase the *plasticity* of the map

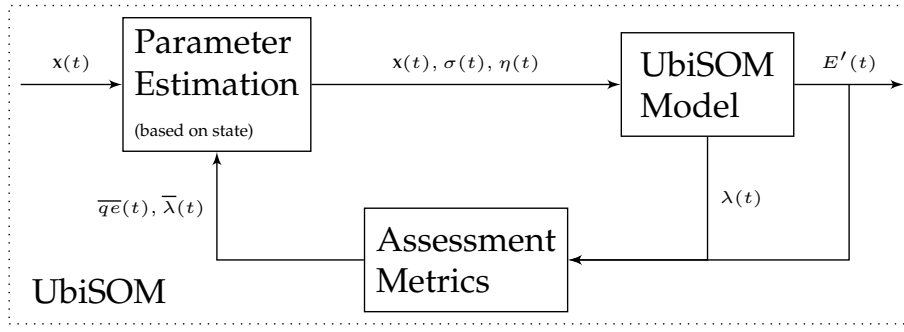


Figure 5.7: The UbiSOM algorithm seen as a feedback control system.

to a point where  $d(t)$  should decrease again. The temporarily increased *plasticity* should allow the map to adjust to the distribution change.

Given that in this state the map is expected to start converging, the values of  $d(t)$  should also decrease. Additionally,  $d(t_f)$  also limits the maximum values that learning parameters can attain during this state. However, there may be cases of abrupt changes from where the map cannot recover, i.e., the map does not resume convergence after increasing  $d(t)$  values. Therefore, if it is detected that learning parameters are in their peak values during at least  $T$  iterations, then this situation is assumed and the UbiSOM transitions back to the ordering state. This transition event is detected using Eq. (5.11).

$$\text{if } \sum 1_{\{d(t) \geq d(t_f)\}} \geq T \rightarrow \text{ordering\_state} \quad (5.11)$$

### Algorithm Formalization

Based on the previous definitions of the assessment metrics, update rule, possible states and correspondent learning parameter estimation methods, the UbiSOM algorithm is formalized in Algorithm 3.

## 5.4 UbiSOM Algorithm Analysis

The UbiSOM is a deterministic algorithm given the same initialization of the map and the same presentation order of observations. Non-determinism can only be introduced by randomly selecting “tied” neurons in the BMU search. However, this is a very rare event and all implementations of SOM algorithms present in this thesis select the first BMU. No evidence was found in literature to favor an approach over the other.

In the *ordering state*, the UbiSOM algorithm behaves in an identical manner to the *Online SOM* algorithm, which has been validated empirically over decades of use. But, the dynamics involved in the *learning state* are more intricate. At a higher level of abstraction, this phase of the UbiSOM algorithm resembles a *feedback control system*. A feedback



**Algorithm 3:** The UbiSOM algorithm.

---

**Input:**  $X$ : A sequence of observations  $\mathbf{x}_t$   
 $\eta_i$ : Initial learning rate for the ordering state  
 $\eta_f$ : Final learning rate for the ordering state  
 $\sigma_i$ : Initial neighborhood radius for the ordering state  
 $\sigma_f$ : Final neighborhood radius for the ordering state  
 $T$ : Window length for assessment metrics  
 $\beta$ : Weight factor for the drift function

**Output:** A codebook  $\mathcal{K}$ , containing  $K$  topologically ordered prototypes

---

```

1 begin
2   Initialize  $\mathcal{W}_k$  with random prototypes and  $t_k^{update} \leftarrow 0$ ;
3    $t \leftarrow 0$ ;
4    $t_i \leftarrow 0$ ;
5    $t_f \leftarrow t_i + T - 1$ ;
6    $current\_state \leftarrow ordering\_state$ ;
7   foreach  $\mathbf{x}_t$  do
8     for  $k \leftarrow 1$  to  $K$  do
9        $\mathbf{w}_c \leftarrow \min_k \|\mathbf{x} - \mathbf{w}_k\|$ ;
10      Compute  $\lambda(t)$  using Eq. (5.3)          /* exploit loop */;
11    end
12     $t_c^{bmu} \leftarrow t$ ;
13    Compute  $E'_q(t)$  using Eq. (5.1);
14    Process  $E'_q(t)$  into  $\bar{q}\bar{e}$  queue;
15    Process  $\lambda(t)$  into  $\bar{\lambda}$  queue;
16    if  $current\_state = ordering\_state$  then
17      Compute values of  $\sigma(t)$  and  $\eta(t)$  using Eq. (5.9);
18    else                                     /* learning state */
19      Compute values of  $\sigma(t)$  and  $\eta(t)$  using Eq. (5.10);
20    end
21    for  $k \leftarrow 1$  to  $K$  do
22      Update prototype  $\mathbf{w}_k$  by (5.8);
23       $t_k^{update} \leftarrow t$ ;
24    end
25     $t \leftarrow t + 1$ ;
26    /* check state transitions                                     */
27    if  $t = t_f$  then
28       $current\_state \leftarrow learning\_state$ ;
29    else if  $\sum 1_{\{d(t) \geq d(t_f)\}} \geq T$  then
30       $current\_state \leftarrow ordering\_state$ ;
31       $t_i \leftarrow t$ ;
32       $t_f \leftarrow t_i + T - 1$ ;
33      Set all  $t_k^{update} \leftarrow t$ ;
34    end
35 end

```

---

control system is a control system which uses the concept of an open loop system as its forward path, but has one or more feedback loops (hence its name) or paths between its output and its input. Effectively, by establishing the inputs  $\mathbf{x}(t)$ ,  $\sigma(t)$  and  $\eta(t)$  to the UbiSOM model, there is a feedback loop in the form of averaged  $E'_q(t)$  and  $\lambda(t)$  values, obtained as outputs. These, in turn, affect some of the next inputs, namely  $\sigma(t)$  and  $\eta(t)$ . Figure 5.7 illustrates this feedback loop. These feedback control systems, also named *closed-loop control systems*, are prevalent in the *Process Control* discipline, monitoring the output of a process and “feeding” some of it back to compare the actual output with the desired output so as to reduce the error; and, if disturbed, bring the output of the system back to the original or desired response. However, the UbiSOM does not explicitly establish a desired response, but relies on the standard dynamics of the SOM to converge during stationary phases of data streams. These dynamics are partially described in the experimental results of Section 5.5.

The UbiSOM algorithm introduces two new parameters, namely  $T$  and  $\beta$ . Regarding  $T$ , it establishes a short, medium or long term trend for the assessment metrics. Also, it establishes the duration of the UbiSOM ordering state. As *Kohonen* recommended, the ordering phase of the SOM should comprise at least the presentation of 1000 observations (*Kohonen 2001*), therefore we can establish  $T \geq 1000$  as a starting point. With respect to  $\beta$ , we should choose values in the upper-half of the  $[0, 1]$  range, so the *average quantization error* metric is favored in the drift function  $d(t)$ . Notwithstanding this empirical argumentation, a parameter sensitivity analysis is presented in Section 5.5.3 to validate these assumptions and to help select good values for these new parameters. Also, besides these two specific UbiSOM parameters, the algorithm also relies on the same parameter-space of the *Online* SOM algorithm for the *ordering* state, namely the parameters  $\{\eta_i, \eta_f, \sigma_i, \sigma_f\}$ . A discussion about the parameterization of the *Online* SOM algorithm can be found in Section 2.4.4 and we should recall that these parameters are set empirically (*Kohonen 2013*). There, the following statement was made: “it is suggested that the width of the neighborhood should be decreased from a width approximately on the order of half of the diameter of the lattice, e.g.,  $\sigma_i = 1/2 \sqrt{(\text{width} - 1)^2 + (\text{height} - 1)^2}$ , for an initial global ordering of the prototypes, down to only encompassing the adjacent neurons, e.g.,  $\sigma_f = 1$ ; the learning rate should, for example, decrease from  $\eta_i = 0.1$  to  $\eta_f = 0.01$ ”. To transpose these suggestions to the UbiSOM requires that we keep in mind two things: first, in the UbiSOM, the  $\sigma(t)$  values are normalized between  $[0, 1]$ , dependent on the largest diagonal of the lattice (illustrative values were depicted in Figure 5.5), and; second, when transitioning to the *learning* state, we must allow the remaining of the learning process to be dependent of the assessment metrics and, consequently, the drift function. That is, we do not want very small values immediately for  $\eta(t)$  and  $\sigma(t)$ , but instead allow their estimation to be governed by the drift function. With this in mind, and after hundreds of experimental tests, the following values were found to work well for all tested data:  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$  and  $\sigma_f = 0.2$ . Regarding the classical SOM algorithms,  $\eta_i$  has the same recommended initial

value as well as  $\sigma_i$ , the later in the sense that it is normalized. Based on Figure 5.5, the value of  $\sigma_i = 0.6$  imposes high updates on the order of half of the diameter of the lattice. The final values of  $\eta_f$  and  $\sigma_f$  are chosen to let the rest of the convergence be dependent on the drift function. For example, in Section 5.5.4 when presenting the evolution of the learning parameters with stationary data streams, it is frequent to see the estimated  $\eta(t)$  and  $\sigma(t)$  values in the learning state attaining values of 0.01 and 0.02, respectively; these later values are comparatively very close to the recommended final values in the classical SOM algorithms. Also, by normalizing  $\sigma(t)$  values, they perform well independently of the lattice size, because the reach of the neighborhood kernel is linearly proportional to the lattice size.

### 5.4.1 Time and Space Complexity

The UbiSOM algorithm does not increase the time complexity of the *Online* SOM algorithm, since all the potentially penalizing additional operations, namely the computations of the assessment metrics, can be performed in  $O(1)$ . It should be noted that the computation of  $\lambda(t)$  values should be performed in parallel with the BMU search.

Regarding space complexity, it increases the space needed for: (i) storing two additional *timestamps*  $T_k^{update}$  and  $T_k^{bmu}$  for each neuron; (ii) storing two queues for the assessment metrics  $\bar{q}e(t)$  and  $\bar{\lambda}(t)$ , each of length  $T$ . Therefore, after the initial creation of data structures (map and queues) in  $O(K)$  time and  $O(Kd + 2K + 2T)$  space, every observation  $\mathbf{x}(t)$  is processed in constant  $O(2Kd)$  time and constant space. No past observations are kept in memory.

Hence, the UbiSOM algorithm is scalable in respect to the number of observations  $N$ , since the cost per observations is kept constant. However, the increase of the number of neurons  $K$ , i.e., the size of the lattice, and the dimensionality  $d$  of the data stream will increase this cost proportionally to  $K$  and/or  $d$ .

## 5.5 UbiSOM Experimental Evaluation

Several experiments were conducted to evaluate the UbiSOM algorithm, using stationary and non-stationary artificial data streams. Again, artificial data was chosen for this purpose so we can establish the ground truth of the expected outcome and illustrate some key points.

### 5.5.1 Experimental Evaluation Overview

The presented experiments focus on parameter sensitivity analysis (PSA) to determine best values for the introduced parameters  $T$  and  $\beta$ ; the impact of these values on the evolution of learning parameters; convergence over stationary and non-stationary data streams; exploratory cluster analysis over resulting maps, and; how the UbiSOM algorithm compares against other variants, namely the Online SOM, PLSOM and DSOM.

Name	$d$	$N$	Data Streams			Type of Evaluation			
			Stationary	Change at	#Clusters	PSA	LPC	ECA	COA
<i>Gauss</i>	2	100 000	Y	-	1	✓+	+		✓
<i>Complex</i>	2	100 000	Y	-	7	✓+	✓	✓	
<i>Chain</i>	3	100 000	Y	-	2	✓+	✓	✓	✓
<i>d5k20</i>	5	300 000	Y	-	20	✓+	+	+	
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2	✓+	+		
<i>DriftTwoOne</i>	2	200 000	N	[50 001, 150 000]	2/1	✓+	+		
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2	✓+	✓	✓	✓
<i>Hepta</i>	3	150 000	N	100 001	7/6	✓+	✓	✓	✓

Table 5.2: Experimental evaluation overview for the UbiSOM algorithm. The symbol ✓ identifies results presented in this chapter; experiments marked with + are either detailed or presented in Appendix C. Parameter sensitivity analysis (PSA); evolution of learning parameters and convergence (LPC); exploratory cluster analysis (ECA), and; comparison with other algorithms (COA).

The artificial data streams used in the present evaluation were briefly described in Section 3.6 and are illustrated in Appendix B. Also, detailed results for the PSA were moved to Appendix C, as well as other results involving particular data streams. Table 5.2 summarizes the conducted experiments, the respective artificial data streams used and where the results can be found.

### 5.5.2 Data Normalization and Algorithm Parameters

All artificial data streams were normalized in the unit hypercube, i.e.,  $\mathbf{x}(t) \in [0, 1]^d$ . Based on the discussion of Section 5.4, the UbiSOM was parameterized with  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$  and  $\sigma_f = 0.2$ , while the remaining parameters  $T$  and  $\beta$  were subjected to a parameter sensitivity analysis in the next section. The chosen size of the lattice was  $20 \times 40$ , as in the previous chapter.

Maps across all experiments and variants use the same random initialization of prototypes at the center of the input space, so no results are affected by different initial states.

### 5.5.3 Parameter sensitivity analysis

This section presents a PSA for parameters  $T$  and  $\beta$  introduced in the UbiSOM. The first establishes the length of the sliding window used to compute the assessment metrics, and consequently whether it uses a short, medium or long-term trend to estimate learning parameters and react to change. While a shorter window is more sensitive to the variance of  $E'_q(t)$  and to noise, a longer window increases the reaction time of the algorithm to true change in the underlying distribution. It also implicitly dictates the duration of the *ordering* state, where *Kohonen* recommends, as a rule-of-thumb, that it should not cover less

that 1000 examples (Kohonen 2001). The second parameter  $\beta$  weighs the importance of both assessment metrics in the drift function  $d(t)$  and, as discussed earlier, we should use higher values so as to favor the  $\overline{qe}(t)$  metric while estimating learning parameters. Hence, the parameter sensitivity analysis was performed in two dimensions for  $T = \{500, 1000, 1500, 2000, 2500, 3000\}$  and  $\beta = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ , as the sets of tested values.

To assess the impact of these parameters, the following values were computed:

**Mean error and neuron activity.** The *mean quantization error* (Mean  $E'_q(t)$  or  $\overline{QE}$ ) characterizes the quality of the quantization procedure along the entire stream, i.e., we should look for lower values (see Section 2.4.6). Similarly, the *mean neuron activity* (Mean  $\lambda(t)$ ) characterizes the neuron usage during learning from stationary and non-stationary data streams, measured between 0 and 1. While it is normal to have some unused neurons separating clusters when projecting the input space, a large portion of unused neurons is undesirable. Consequently, we should look for higher values of this measure. Both measurements must not be confused with the assessment metrics  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$ , since they average values within a sliding window.

**Topographic Error.** The *mean topographic error* (Mean  $TE(t)$  or  $\overline{TE}$ ) measures undesirable distortions of the map, i.e., topological defects (see Sections 2.4.4 and 2.4.6). Values greater than zero may not indicate topological defects, but should remain fairly close to zero.

**Convergence time.** The assessment metric  $\overline{qe}(t)$ , for the different values of  $T$ , was used to obtain a grasp on the delay in convergence imposed by this parameter. From the minimum  $\overline{qe}(t)$  obtained throughout the stream, the iteration where the  $\overline{qe}(t)$  value falls within 5% of the overall minimum was computed (Convergence  $t$ ), as a temporal indicator of convergence. In other words, this value indicates at which point in time the map attained a convergence of 95% against the minimum  $\overline{qe}(t)$  obtained throughout the data stream. This value is merely indicative and should be regarded with care, as it is a hard comparison, i.e., a value very close can be obtained much earlier, so it is not one of the main quantitative results we are interested in. Also, with non-stationary data streams, it may not be very meaningful.

**Number of resets.** The number of transitions back to the *ordering state* (resets) was also obtained. A transition to this state indicates that the UbiSOM algorithm was unable to adapt to the underlying distribution with the established learning parameters thresholds inherited from the ordering state, i.e.,  $\eta_f$  and  $\sigma_f$ . Ideally, during the presentation of the artificial data streams, we want the least number of resets; this indicates that the established parameters allowed the UbiSOM to properly adjust itself to any changes in the underlying data stream. This metric is relevant for non-stationary data streams.

Regarding the PSA only  $Mean E'_q(t)$ ,  $Mean \lambda(t)$  and  $Mean TE(t)$  are used, while the remaining are merely indicative. Thus, the goal was to find the best parameters for each problem (data stream) that could simultaneously minimize the mean quantization error and mean topographic error, while maximizing the mean neuron utility. This can be regarded as a multi-objective optimization problem (Marler and Arora 2004). Let  $f_E(T, \beta)$ ,  $f_\lambda(T, \beta)$  and  $f_{TE}(T, \beta)$  be the objective functions, i.e., values obtained for  $Mean E'_q(t)$ ,  $Mean \lambda(t)$  and  $Mean TE(t)$ , respectively, for a particular parameterization. The best parameters for a particular data stream are obtained by maximization of the function  $g(T, \beta) = f_\lambda(T, \beta) - f_{QE}(T, \beta) - f_{TE}(T, \beta)$ . For this to hold, all objective functions should be dimensionless, which is achieved through normalization. The following Eq. (5.12) establishes the *optimization function* (criteria) used to derive the best parameters for each data stream:

$$\underset{T, \beta}{\text{maximize}} \quad g(T, \beta) = \frac{f_\lambda(T, \beta)}{f_\lambda^{max}} - \frac{f_E(T, \beta)}{f_E^{max}} - \frac{f_{TE}(T, \beta)}{f_{TE}^{max}} \quad . \quad (5.12)$$

This results in a non-dimensional optimization function with an upper limit of one and an unbounded lower limit (please note that  $f_i^{max} \neq 0$  is assumed).

Table 5.3 summarizes the PSA for all data streams and reveals the best three parameterizations found by the multi-objective optimization procedure. For each data stream, the top three parameterizations are shown, by decreasing order of the obtained values for Eq. (5.12). Full individual results can be found in Appendix C, where the individual 10 best values for each objective function (metric) are highlighted. This highlighting was also transposed to Table 5.3.

## Findings

Any algorithm has a set of optimal parameters for a particular problem. This, obviously, also applies to the UbiSOM algorithm and it can be seen throughout the different best parameterizations found across all data streams. Notwithstanding this fact, there are some conclusions we can derive from the PSA results that will further allow us to devise some good intervals of values for the parameters  $T$  and  $\beta$ . From the obtained procedure and results, we can conclude the following:

- Objective functions translate competing goals. From the results it is clear that no obtained parameterization yields the best values across the three optimization criteria. Hence, the optimization procedure finds the best compromise solution in the parameter-space. The competing goals subject has already been addressed in Section 2.4.6, when discussing quality assessment metrics of the SOM. For example, given the UbiSOM performs density matching, when converging a region of prototypes to a *Gaussian* shaped cluster, there will forcefully be some kind of warping of the lattice to concentrate more neurons in the denser region; therefore, we cannot

Data stream	$T$	$\beta$	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean $TE(t)$	Convergence $t$	Resets
<i>Gauss</i>	1500	0.5-1.0	7.50597e-03	1	7.960e-03	14 314	0
<i>Complex</i>	1500	0.8	9.404490e-03	9.919812e-01	6.890e-03	9 209	0
	1500	0.9	9.436820e-03	9.842766e-01	6.950e-03	9 197	0
	2500	0.8	1.016753e-02	9.934998e-01	5.750e-03	9 855	0
<i>Chain</i>	1000	0.6	1.578697e-02	9.953658e-01	3.365e-02	7 509	0
	1000	0.5	1.575886e-02	9.966253e-01	3.425e-02	7 029	0
	1000	0.7	1.577986e-02	9.935471e-01	3.438e-02	7 480	0
<i>d5k20</i>	2000	0.7	2.025618e-02	9.416751e-01	8.580000e-03	5 842	0
	1500	0.7	2.179408e-02	9.379426e-01	4.513330e-03	4 530	0
	1000	0.7	2.258051e-02	9.359845e-01	6.583330e-03	3 559	0
<i>AbruptOneTwo</i>	3000	0.5-1.0	5.36115e-03	1	5.35e-03	13 091	0
<i>DriftTwoOne</i>	1000	0.8	4.85912e-03	9.927826e-01	4.0850e-03	27 884	0
	1000	0.7	4.89365e-03	9.958548e-01	4.0800e-03	25 809	0
	1500	0.8	4.82665e-03	9.929180e-01	4.3900e-03	26 214	0
<i>Clouds</i>	2500	0.6	5.26397e-03	9.981086e-01	1.243031e-02	10 176	1
	2000	0.5	5.23318e-03	9.988502e-01	1.400535e-02	9 794	1
	1000	0.6	5.00901e-03	9.976649e-01	1.548039e-02	7 161	1
<i>Hepta</i>	500	0.6	1.391013e-02	9.917042e-01	3.583333e-02	8 852	0
	500	0.5	1.398475e-02	9.945278e-01	3.646000e-02	8 843	0
	500	0.7	1.372421e-02	9.861660e-01	3.832000e-02	8 866	0

Table 5.3: Summary of the UbiSOM parameter sensitivity analysis. For each data stream the three best (descending order) parameterizations found are shown.



attain a perfect score in the  $\overline{TE}$  metric and allow at the same time a near optimal VQ procedure. Also and frequently, there are situations where the “tension” of the neighboring relations will necessarily lead to some unrepresentative prototypes after convergence, which affect the Mean  $\lambda(t)$  metric. Moreover, each objective function has the same overall weight in the optimization procedure. Those weights can be changed, but there is no clear way of choosing the particular weights. Hence, we assume they all have equal importance in determining the quality of the obtained maps;

- The results clearly validate the use of the average neuron utility  $\overline{\lambda}(t)$  metric, since all best parameterizations found dictate that  $\beta < 1$  — please recall that if  $\beta = 1$ , then only the average quantization error  $\overline{qe}(t)$  metric is used;
- Therefore, regarding parameter  $\beta$ :
  - All best solutions involve  $\beta \geq 0.5$ , also validating the previous empirical assumptions that the average quantization error should be favored when weighing the drift function;
  - In general, no best compromise solution is obtained with  $\beta = 1$ . Although this could be expected for non-stationary data streams, as a means to detect obsolete regions of neurons, the same also applies to stationary data streams. Results show that solely using the average quantization error does not achieve the best mean quantization error along the data stream, even with stationary distributions. This clearly validates the use of the average neuron activity assessment metric and is a very interesting result, in the sense that it can help produce better maps in terms of VQ. This is later confirmed in the results of Section 5.5.4, where evolutions of learning parameters are depicted;
  - Moreover, results for both stationary and non-stationary data streams suggest the interval  $\beta \in [0.6, 0.9]$ .
- Regarding parameter  $T$ :
  - In general, the best solutions are obtained effectively by using  $T \geq 1000$ . This is also an interesting result that is in line with *Kohonen's* suggestion of an ordering phase not shorter than 1000 observations;
  - There is no clear differentiation in  $T$  values across stationary and non-stationary data streams, an indication that indeed there is an optimal value for each problem. However, results suggest that  $T \in [1000, 2500]$  is a good interval.
  - As a secondary observation, it comes at no surprise that for increasing  $T$ , the convergence (*Convergent*) happens latter in the stream. This is because the ordering state is extended in time and the assessment metrics are computing a longer term trend.



- Finally, in terms of necessary *resets*, only the *Clouds* data stream required a reset in the UbiSOM algorithm. This is later depicted and discussed in Section 5.5.4.

Regarding particular results, the *Gauss* data stream always indicates a  $\bar{\lambda}(t) \approx 1$ , because the whole map is used to abstract a single *Gaussian* cloud. This also happens with the *AbruptOneTwo* data stream, composed by one/two *Gaussian* clouds.

The most important thing that must be taken into consideration is that the UbiSOM performs well against the above data streams with a variety of parameterizations, not only the best found through the PSA. Also, in some particular cases that will be addressed, other parameterization may be better than the one found in the PSA. This is due to the equal importance given to each objective function. Depending on the distributions, it may be better to allow a worst neuron utility, favoring a better quantization value. One final result obtained during other experimental tests, regarding the above discussion, is that the parameterization  $T = 2000$  and  $\beta = 0.8$  performed well across all tested data streams. These parameters are, coincidentally, in the middle of the suggested intervals.

#### 5.5.4 Convergence with stationary and non-stationary data

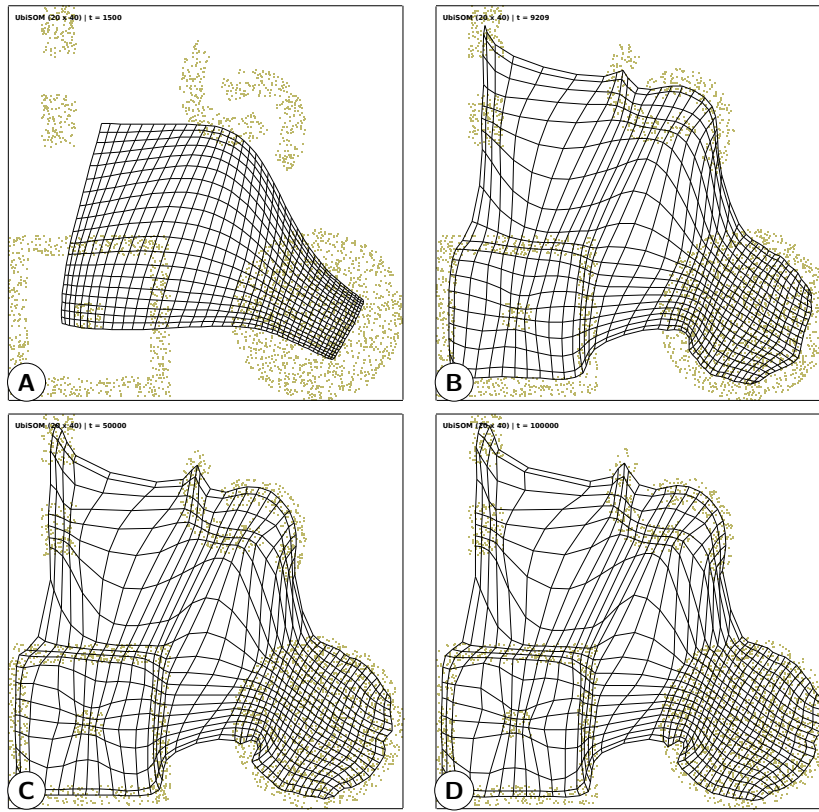
This section focuses on four artificial data streams, namely the *Complex*, *Chain*, *Clouds* and *Hepta* data streams to illustrate the behavior of the UbiSOM algorithm and the evolution of the learning parameters. Results for some other artificial data streams are presented in Appendix C, as depicted in Table 5.2. The aforementioned data streams were chosen to illustrate some key points of the UbiSOM algorithm with stationary and non-stationary data. The parameters  $T$  and  $\beta$  used for each data stream were the best obtained from the PSA, presented in Table 5.3 (namely, in the top respective rows).

For each data stream, the UbiSOM lattice is shown at specific instants of the data stream, depicting how the map evolves over time. Also, obtained values for the local quantization error  $E'_q(t)$ , assessment metrics  $\overline{qe}(t)$  and  $\bar{\lambda}(t)$ , and learning parameters  $\eta(t)$  and  $\sigma(t)$  are shown, depicting the evolution of the learning procedure. The drift function  $d(t)$  is not explicitly shown, since the learning parameters are estimated proportionally to it. Hence, the behavior of drift function  $d(t)$  is the same as of  $\eta(t)$  and  $\sigma(t)$  during the *learning state*, i.e., a weighted average between  $\overline{qe}(t)$  and  $\bar{\lambda}(t)$ .

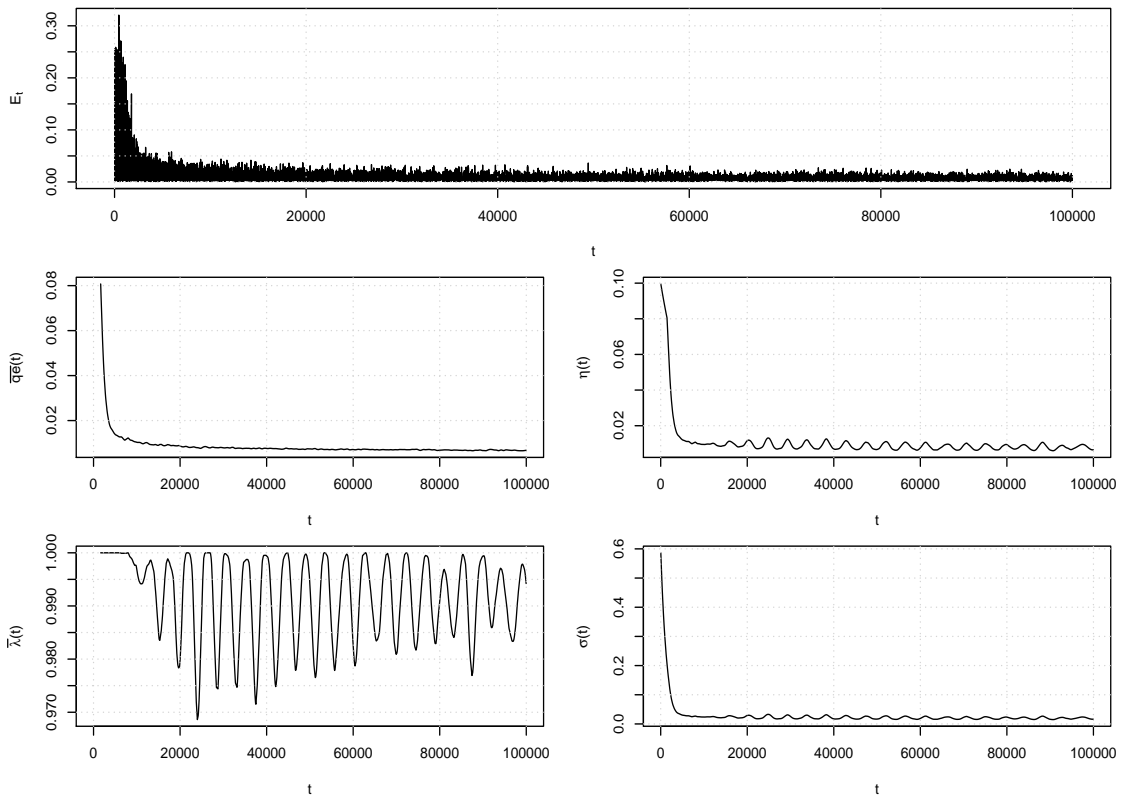
#### Stationary Data Streams

The following artificial data streams illustrate the behavior of the UbiSOM over stationary data, namely the *Complex* and *Chain* data streams. Given the similarity of the results, both are interpreted together.

**Complex** Figure 5.8 depicts the evolution of the UbiSOM with  $T = 1500$  and  $\beta = 0.8$  over the *Complex* data stream, illustrating the map at  $t = \{1500, 9209, 50\,000, 100\,000\}$ .



(a) Convergence of the map along time: (A)  $t = 1500$ ; (B)  $t = 9209$ ; (C)  $t = 50\,000$ , and; (D)  $t = 100\,000$ .



(b) Assessment metrics and learning parameters evolution.

Figure 5.8: The UbiSOM evolution over the stationary *Complex* data stream, with  $T = 1500$  and  $\beta = 0.8$ .

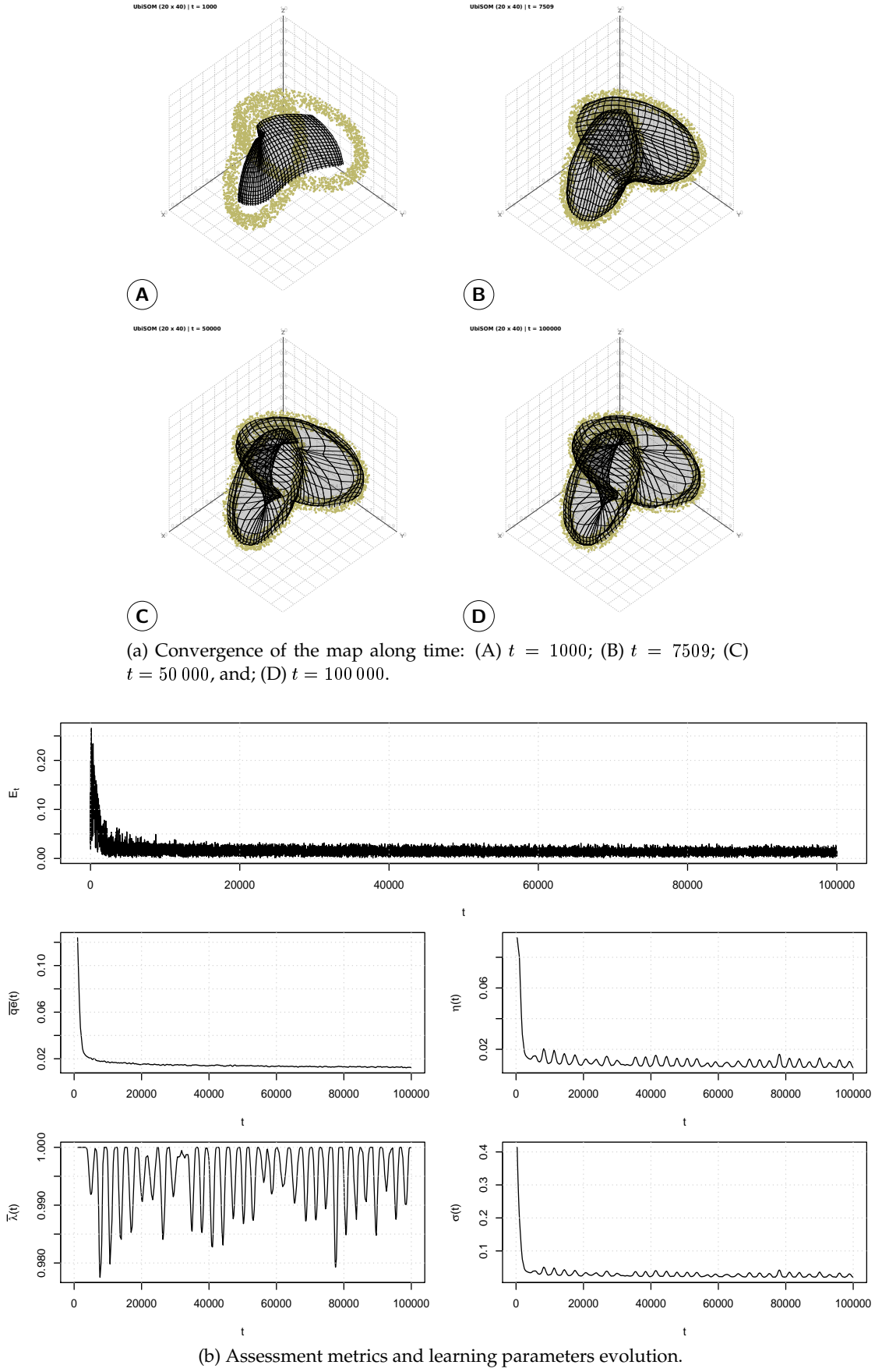
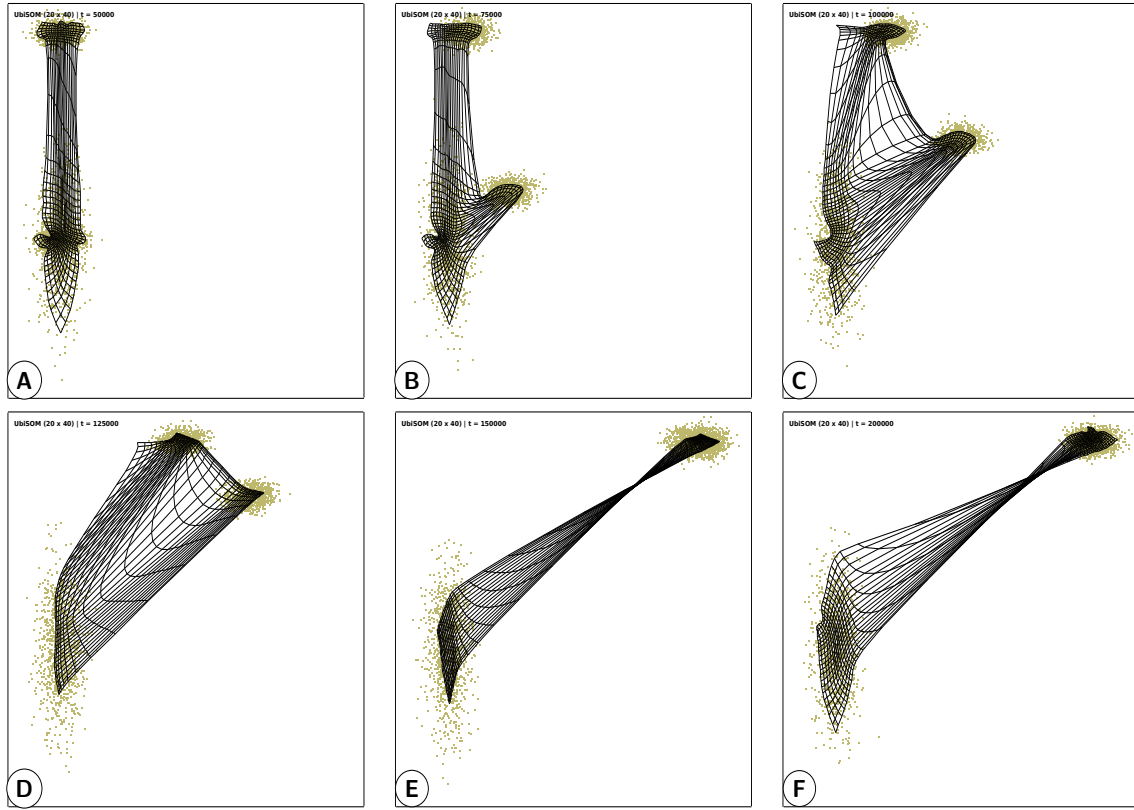
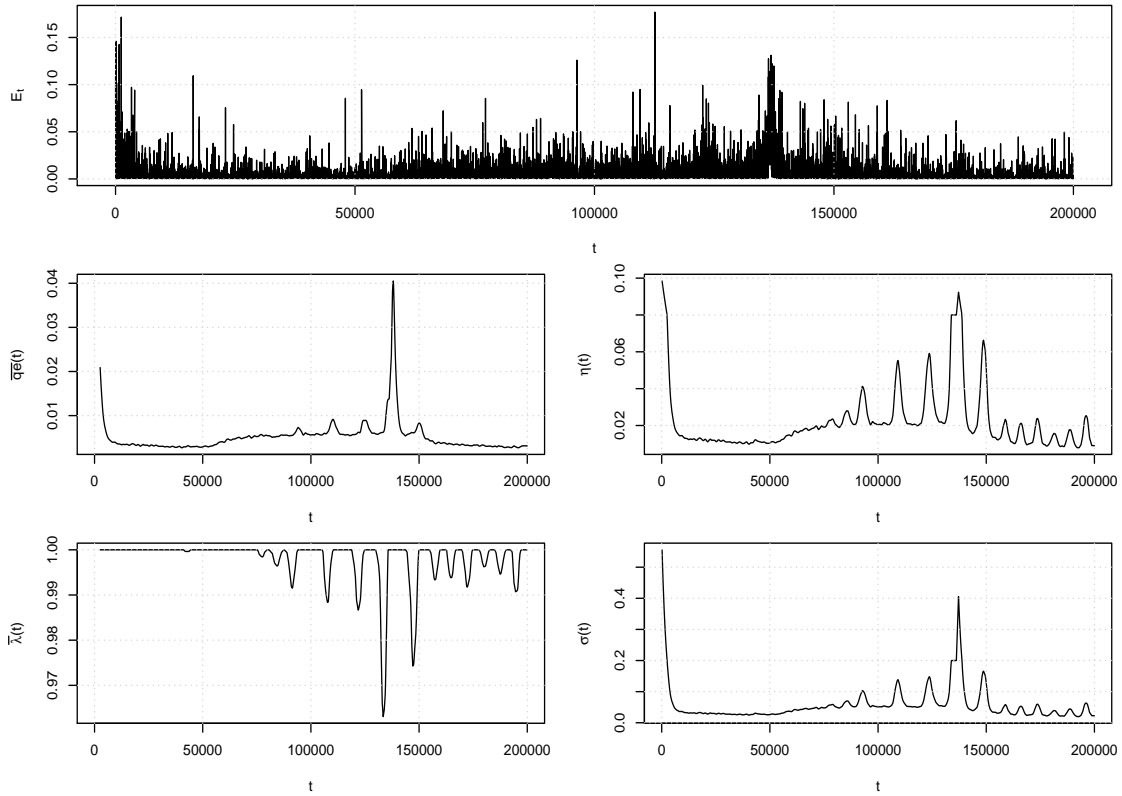


Figure 5.9: The UbiSOM evolution over the stationary *Chain* data stream, with  $T = 1000$  and  $\beta = 0.6$ .



(a) Convergence of the map along time, where a *reset* occurred at  $t = 136\,295$ : (A)  $t = 50\,000$ ; (B)  $t = 75\,000$ ; (C)  $t = 100\,000$ ; (D)  $t = 125\,000$ ; (E)  $t = 150\,000$ , and; (F)  $t = 200\,000$ .



(b) Assessment metrics and learning parameters evolution.

Figure 5.10: The UbiSOM evolution over the stationary *Clouds* data stream, with  $T = 2500$  and  $\beta = 0.6$ .

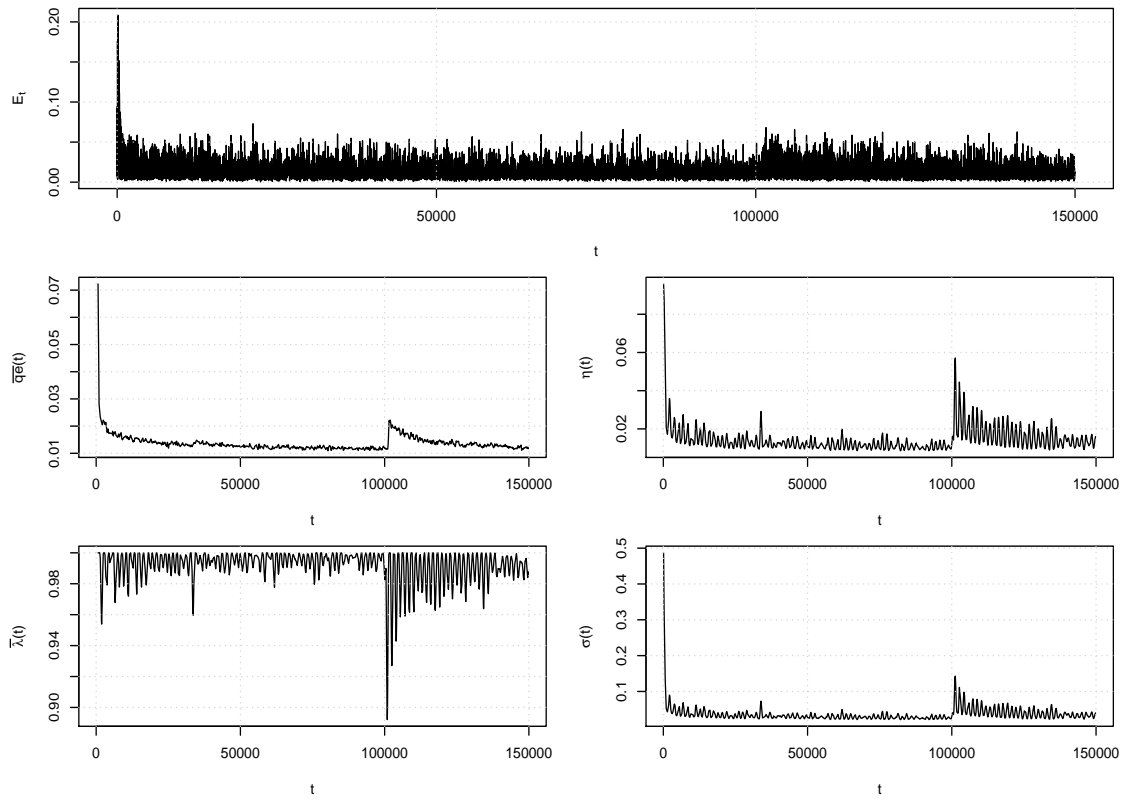
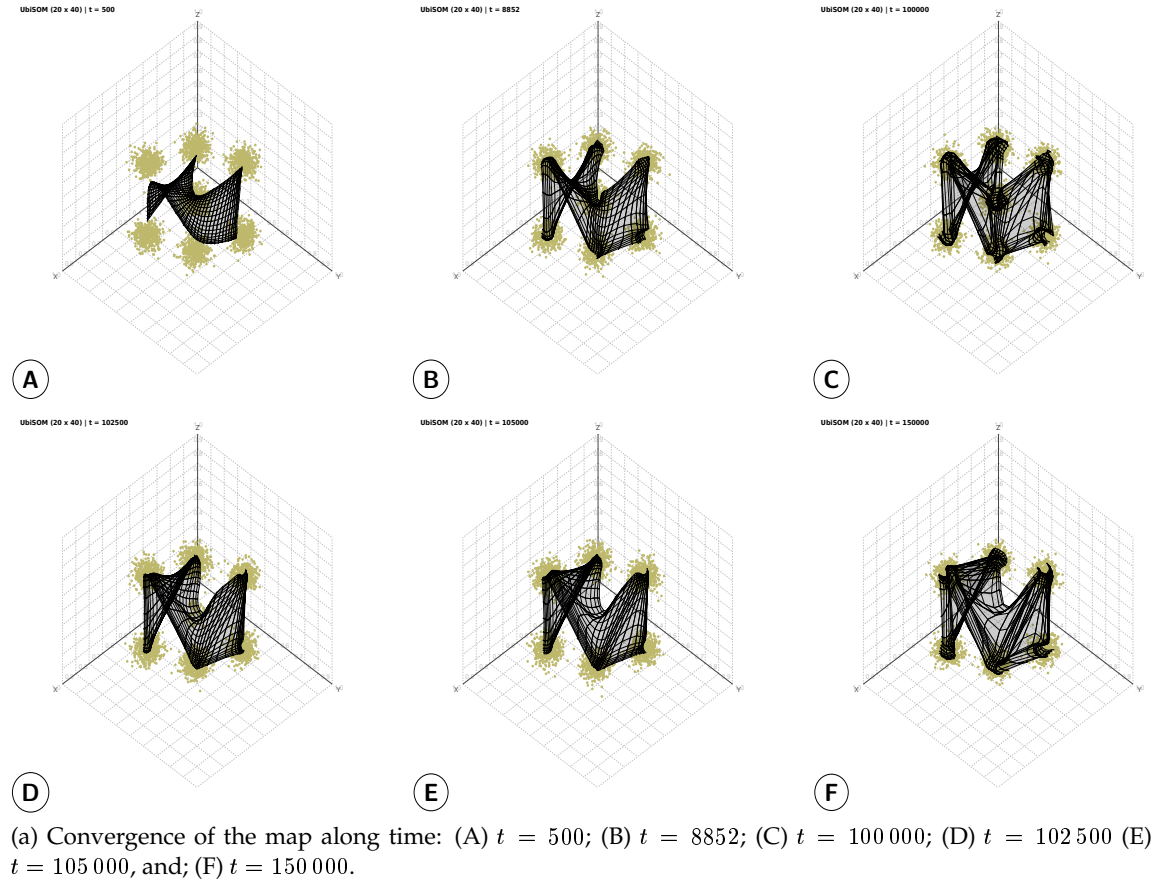


Figure 5.11: The UbiSOM evolution over the stationary *Hepta* data stream, with  $T = 500$  and  $\beta = 0.6$ .

**Chain** Similarly, Figure 5.9 depicts the UbiSOM over the *Chain* data stream with  $T = 1000$  and  $\beta = 0.6$ , illustrating the map at  $t = \{1000, 7509, 50\,000, 100\,000\}$ .

Obtained maps are illustrated in Figures 5.8a and 5.9a, at specific points in time: (A) when the algorithm transitions to the *learning state*; (B) at *Convergence*  $t$  from the PSA; (C) at the middle of the data stream, and; (D) at the end of the data stream. In (A) it is visible that both maps have unfolded correctly over the distribution with no topological defects. However, the maps have not yet converged sufficiently over the distribution, deterring any meaningful use of exploratory visualizations at this time. However, after (B) the maps achieved approximately 95% of convergence, where in (C) appear to already achieved a stable and full convergence state, since there are no significant visible changes regarding (D). By inspecting the  $\overline{qe}(t)$  evolution, indeed after  $t \approx 20\,000$  this assessment metric does not improve greatly in both data streams.

Figures 5.8b and 5.9b depict the evolution of the assessment metrics and learning parameters. Firstly, one should recall that  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$  values are only available when the algorithm transitions to the learning state. Consequently, it can be seen that  $\overline{qe}(t)$  decreases rapidly after the ordering state. This favors the assumption that the *average quantization error* decreases when the distribution is stationary. Also, the fast decreasing rate is interesting, which seems to evidenciate that the UbiSOM algorithm achieves a sort of, absent of a better term, “natural” convergence over the distribution. This behavior is consistent throughout all tested stationary data streams and during stationary phases of non-stationary data streams, given appropriate parameterization.

Another interesting result is the behavior of  $\overline{\lambda}(t)$ , given these somewhat complex cluster structures are prone to the existence of “dead-units” between clusters. When maps start to properly “cover” the distributions, there are neurons that begin to not be selected as BMUs, translating to values  $\lambda(t) < 1$ . As the  $\overline{\lambda}(t)$  metric reacts with a decreasing trend, both learning parameters are increased proportionally (note that  $\overline{qe}(t)$  is approaching stable values). This in turn, by increasing the learning parameters, namely the neighborhood radius, increases  $\overline{\lambda}(t)$ . Hence, this mutual dependence generates a “wave” pattern in the learning parameters. However, please note that  $\overline{qe}(t)$  continues to decrease slowly over time. In consideration of the fact that, even with stationary distributions, the PSA analysis did not select any parameterization with  $\beta = 1$ , this indicates that  $\overline{\lambda}(t)$  is valuable even in stationary phases of the data stream, acting as a *conscience* mechanism to utilize as much neurons as possible.

### Non-stationary Data Streams

The following artificial data streams illustrate the behavior of the UbiSOM over non-stationary data. The *Clouds* data stream was chosen to validate and illustrate how the UbiSOM behaves over an evolving distribution, highlighting some dynamics between the assessment metrics and learning parameters. The *Hepta* data stream was chosen to



illustrate how the UbiSOM reacts to the disappearance of a cluster by analyzing the response of the assessment metrics and learning parameters.

**Clouds** Figure 5.10 depicts the evolution of the UbiSOM with  $T = 2500$  and  $\beta = 0.6$  over the *Clouds* data stream, illustrating the map at  $t = \{50\,000, 75\,000, 100\,000, 125\,000, 150\,000, 200\,000\}$ . These times were chosen to illustrate the evolution of the UbiSOM lattice during the non-stationary phase of the data stream, namely during  $t = [50\,000, 150\,000]$ , and the final map obtained.

Regarding Figure 5.10, in (A) we can observe that the UbiSOM lattice is covering both clusters, assigning more neurons to the denser areas, as expected. In (B), (C) and (D), as the bottom cluster splits and the top cluster moves, the lattice is able to follow them correctly and maintain the density matching. In (E), at the end of the non-stationary phase of the data stream, we can observe a “twisted” lattice that further converges until (F).

This behavior can be explained by interpreting the assessment metrics and learning parameters evolution in Figure 5.10b. After initial convergence, until the map depicted in (A), changes start occurring at  $t = 50\,000$ . Soon after,  $\overline{qe}(t)$  increases while  $\overline{\lambda}(t) \approx 1$ , explained by the fact that a “new” cluster will forcibly lead to an increase of the *average quantization error*, since those new observations are not represented adequately by the current codebook; by itself, this leads to increasing learning parameters which allow the map to adjust to these new observations, while maintaining a high neuron utility. However, as the distribution changes continue, the learning parameters do not have sufficient high values to compensate those changes. Consequently, some neurons become stagnant, since they represent old observations. This is reflected in the average neuron utility  $\overline{\lambda}(t)$  falling off periodically after  $t \approx 75\,000$ , causing spike increases in the learning parameters; these, in turn, allow the map to gradually recover.

Regarding the “twisted” lattice, we can observe that the UbiSOM transitioned back to the *ordering state* (at  $t = 136\,295$ ), forcing a reordering of the prototypes in the middle of the distribution change. This highlights two things:

- i. With the current parameterization the learning parameters cannot attain sufficient high values to cope with changes. This is expressed by the spike increase of  $\overline{\lambda}(t)$  and  $\overline{qe}(t)$  afterwards, leading to maximal learning parameters for the *learning state*. The UbiSOM, in this case, is unable to resume convergence and the learning parameters stay maximal. This situation is detected by Eq. (5.11), forcing the UbiSOM to transition back to the *ordering state*;
- ii. The UbiSOM is able to quickly reorder the prototypes, despite changes occurring in the distribution after  $t = 136\,295$ , and converge to the final distribution.

Concerning this *reset* of the UbiSOM, the PSA analysis shows that other parameterizations do not result in a reset during changes in the underlying distribution.

However, some may not allow the map to properly follow changes, while others may, at the cost of a worst  $\overline{QE}$  — this was observed experimentally. Considering this parameterization was the best found in the PSA, one should be able to justify the *reset*: e.g., if we look at the final distribution, the obtained final configuration induces less “stress” on the edges of the lattice and maximizes the neuron utility. Whether the result is considered a topological defect is debatable in this situation. The derived *U-Matrices* presented in the next section indicate that in this case there are no consequences in terms of exploratory cluster analysis.

**Hepta** Figure 5.11 depicts the UbiSOM over the *Hepta* data stream with  $T = 500$  and  $\beta = 0.6$ , illustrating the map at  $t = \{500, 8852, 100\,000, 102\,500, 105\,000, 150\,000\}$ . These times show the UbiSOM lattice right after the transition to the learning state; the lattice at *Convergence*  $t$  from Table 5.3, and; how the UbiSOM reacts to the disappearance of a cluster at  $t = 100\,000$ . Until this point in time the data stream is stationary and the UbiSOM behaves similarly as with the *Complex* and *Chain* data streams; but, given that  $T$  is smaller, the “wave” pattern of  $\bar{\lambda}(t)$  has a higher frequency.

Regarding Figure 5.11b, when the inner cluster disappears (at  $t = 100\,000$ ), the  $\bar{\lambda}(t)$  metric abruptly decreases and increases the learning parameters to values that allow the UbiSOM to adjust itself to the new distribution. By close inspection, the change was not detected by  $\overline{qe}(t)$  at that time, because the remaining clusters are still being quantized the same way as before. In this case the increase of  $\overline{qe}(t)$  is only justified by the increase of the learning parameters, which leads to prototype updates of higher magnitude. This, consequently, temporarily degrades the previous quantization during the adaptation phase. Thus, by observing lattices (D) through (F) in Figure 5.11a, we can see that the UbiSOM was able to adjust itself to the disappearance of the cluster, by comparison with lattices (A) through (C). This was the initial example that led to the proposal of the average neuron utility assessment metric.

One important exception to note, regarding the PSA results, involves the non-stationary *DriftTwoOne* data stream, where PSA results suggest parameterizations that were experimentally found not to be the best. This data stream is a very hard problem for the UbiSOM, because there is a gradual drift in the distribution where two *Gaussian* clusters are merged into one. The best parameters found experimentally are  $T = 1000$  (the same as in the PSA), but  $\beta = 0.5$ , lower than the suggested  $\{0.7, 0.8\}$  values. Only using the lower value, increasing the importance of the average neuron utility metric, can the UbiSOM produce the expected result at the end of the data stream. The comparison between the two results can be found in Appendix C, namely in Section C.2.3 and precisely in Figures C.6 and C.7, respectively. However, the results obtained in the PSA are easily justifiable: by using a relatively low  $\beta$  value, although increasing the responsiveness to the moving



clusters, this causes the lattice to perform “jumps” over this evolving distribution, harming the quantization procedure, and therefore, the quantization errors. Hence, the PSA still upholds its merits regarding the error metrics used in its application.

### 5.5.5 Exploratory Cluster Analysis

In this section we will focus on exploratory cluster analysis, using the *U-Matrix* visualization (see Section 2.4.5), from the previous four data streams. Note that some artificial data streams contain cluster structures that evolve over time. An additional result pertaining the *d5k20* data stream can be found in Appendix C, namely in Figure C.8.

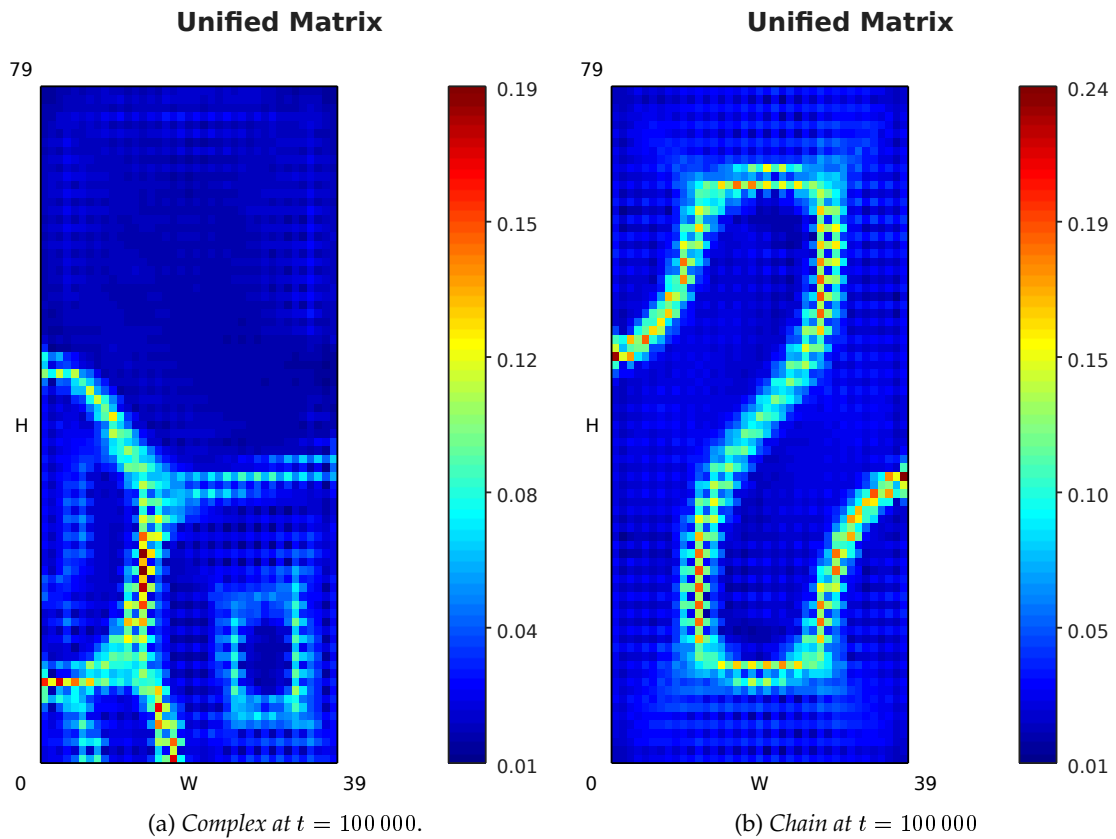


Figure 5.12: U-Matrices obtained for stationary data streams.

**Complex and Chain** Regarding the stationary *Complex* and *Chain* data streams, the corresponding *U-Matrices* of maps obtained at the end of the streams are depicted in Figures 5.12a and 5.12b, respectively. In the *Complex* U-Matrix 6 clear clusters are visible, while a more careful analysis allows us to derive that the one at the center-left represents, in fact, two clusters, summing up to the expected 7 clusters present in this stationary distribution. With respect to the *Chain* data stream the U-Matrix clearly indicates 2 clusters, corresponding to the two interlocked rings.

Concerning the non-stationary *Clouds* and *Hepta* data streams, the U-matrix visualization was applied to maps at different instants of the streams (corresponding to previous instants in Figures 5.10 and 5.11):

**Clouds** Changes in an evolving distribution may be inferred by sequential analysis of the U-Matrices. Figure 5.13a depicts the U-Matrices obtained at  $t = \{50\,000, 75\,000, 100\,000, 125\,000, 150\,000, 200\,000\}$ : (A) depicts 2 clusters of different densities, since more neurons are assigned to the top depicted cluster, corresponding to the bottom-left cluster in the input space (please note that the orientation of the U-Matrix may not match that of the distribution; this can only be inferred together with the *component planes*). In (B) and (C) we can observe splitting of clusters with a new cluster emerging from the denser one (3 clusters). In (D) it is harder to visually infer the three clusters. Here a 3D visualization or *log-normalization* of the U-Matrix values could enhance that tenuous cluster border at the center-left. Finally (E) and (F) depict the final cluster structure consisting of 2 clusters of similar density. Here, after a *reset* of the UbiSOM one loses the relative position of the clusters, since the rotation of the lattice is different from the previous. Please note that the two moving clusters were merged forming a smaller, but denser, cluster in respect to the stationary cluster, hence the similarity in size depicted in the U-matrix.

**Hepta** Figure 5.13b depicts a sequence of U-matrices obtained at  $t = \{8852, 100\,000, 105\,000, 150\,000\}$ : In (A), after 95% of convergence, the 7 clusters are visible. This also holds in (B), but with increased convergence of the map. In (C) only 6 clear clusters are visible, conforming to the disappearance of a cluster; similarly, with increased convergence of the map, the cluster structure is more refined in (D).

### 5.5.6 Comparison against other variants

The UbiSOM algorithm was compared to the *Online SOM* algorithm (since in these experiments the size of the data stream is known), *Parameter-Less SOM* (PLSOM) and *Dynamic SOM* (DSOM) in terms of density matching, convergence with stationary and non-stationary data streams and exploratory cluster analysis from resulting maps (PLSOM and DSOM were described in Section 2.5.2). For these comparisons the *Gauss*, *Chain*, *Clouds* and *Hepta* artificial data streams were chosen to highlight some key results between these variants.

#### Algorithms Parameterization

The UbiSOM parameters remain as  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$  and  $\sigma_f = 0.2$  and the other parameters were set based on the results of the PSA analysis, as before. In summary, *Gauss*:  $\{T = 1500, \beta = 0.7\}$ ; *Chain*:  $\{T = 1000, \beta = 0.6\}$ ; *Clouds*:  $\{T = 1500, \beta = 0.7\}$ , and; *Hepta*:  $\{T = 1500, \beta = 0.7\}$ .

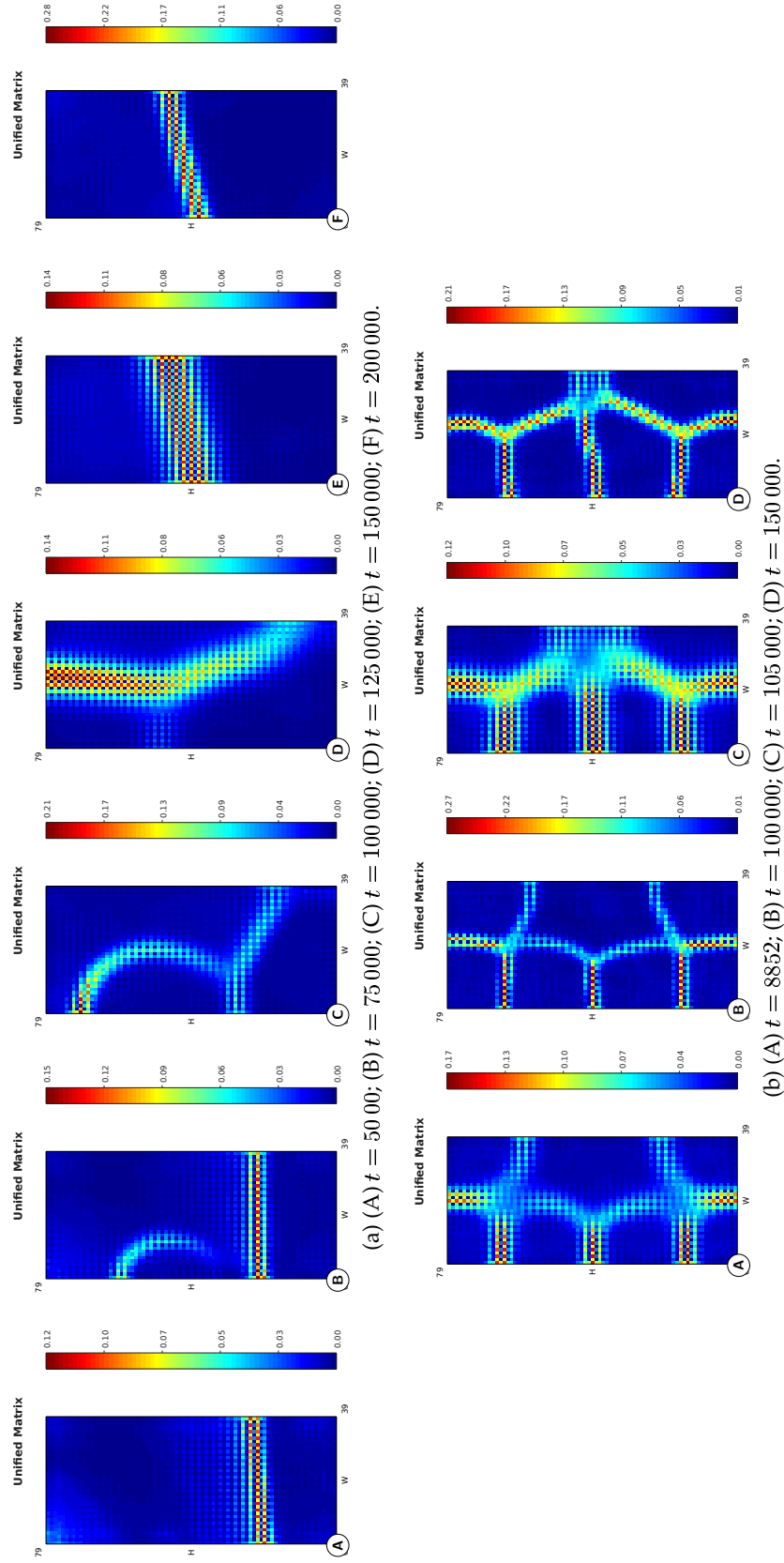


Figure 5.13: U-Matrices obtained for non-stationary data streams: a) *Clouds*, and; b) *Hepta*.

Regarding the other algorithms, after several experiments the best parameters for the chosen map size and for each compared algorithm were selected. The *Online* SOM uses  $\eta_i = 0.1$  and  $\sigma_i = 1/2\sqrt{(width-1)^2 + (height-1)^2}$ , decreasing monotonically to  $\eta_f = 0.01$  and  $\sigma_f = 1$  respectively; PLSOM uses a single parameter  $\gamma$  called *neighborhood range* and the values yielding the best results for the used lattice size were  $\gamma = \{65, 37, 130, 35\}$  for the *Gauss*, *Chain*, *Clouds* and *Hepta* data streams, respectively. DSOM has two constant parameters: the learning rate  $\eta$  and plasticity  $\epsilon$ . Authors in (Rougier and Boniface 2011) use  $\epsilon = 3$  and  $\eta = 0.1$  for a 2D Gaussian distribution, but with the map already spread across the input space. With randomly initialized prototypes, such small values deter the lattice from ever unfolding. This caused some concerns with this variant. Larger values were tried with more success, but its the variant that takes longer to properly unfold the map, usually in the order of 10 000 observations, from a completely random initialization of prototypes and exhibits some topological defects. The best experimentally derived parameters for DSOM were  $\epsilon = \{80, 50, 150, 50\}$  and  $\eta = \{0.55, 0.95, 0.95, 0.65\}$  for the *Gauss*, *Chain*, *Clouds* and *Hepta* data streams, respectively.

### Density Matching

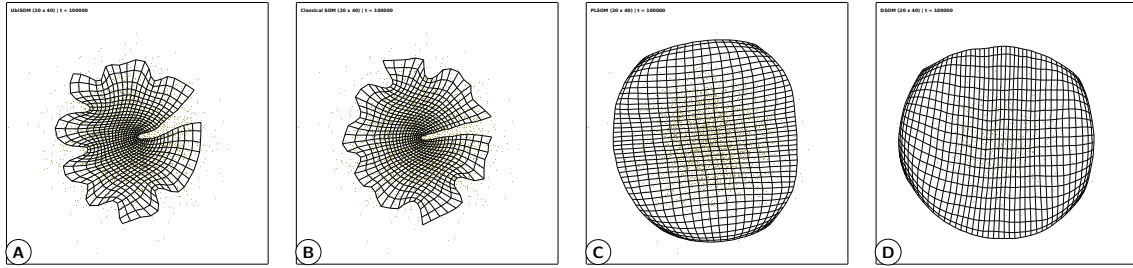


Figure 5.14: Final maps of different SOM variants obtained for the stationary Gaussian data stream: (A) UbiSOM; (B) Online SOM; (C) PLSOM; (D) DSOM.

The obtained models are illustrated for all tested algorithms in Figure 5.14, using the stationary *Gauss* data stream. It can be seen that only the *Online* SOM and the UbiSOM are able to model the input space density correctly, assigning more neurons to the denser area of the input space. The inability of both PLSOM and DSOM to properly map the density through a good VQ may hinder the visualization aspects of the map and exploratory visual discovery in more complex distributions. This is due to only consider the local quantization error  $E'_q(t)$  while estimating learning parameters, which does not allow to monotonically decrease the parameters during stationary states. What is actually mapped by these proposals is the structure or support of the distribution rather than the density, as discussed in Section 2.5.2. Therefore, the UbiSOM is the only variant that can replicate the original SOM results.

Data stream	Algorithm	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean $TE(t)$	Opt. Func. — Eq. (5.12)
<i>Gauss</i>	UbiSOM	7.5060e−3	1	7.9600e−3	−4.3284e−1
	Online	1.4045e−2	9.9988e−1	5.2300e−3	−5.9041e−1
	PLSOM	7.9702e−3	9.6830e−1	8.8600e−3	5.9917e−1
<i>Chain</i>	UbiSOM	1.5787e−2	9.9537e−1	3.3650e−2	9.6655e−2
	Online	3.3551e−2	9.9681e−1	3.2330e−2	−4.1457e−1
	PLSOM	1.4290e−2	1	7.8590e−2	−4.2592e−1
<i>Clouds</i>	UbiSOM	5.2640e−3	9.8109e−1	1.2430e−2	1.4177e−1
	Online	7.3265e−3	8.9579e−1	1.7870e−2	−2.8783e−1
	PLSOM	4.1127e−3	9.4402e−1	8.8955e−2	−5.9913e−1
<i>Hepta</i>	UbiSOM	1.3910e−2	9.9170e−1	3.5833e−2	−2.3273e−1
	Online	2.3251e−2	9.5612e−1	3.2393e−2	−6.0994e−1
	PLSOM	1.2725e−2	9.9979e−1	5.7206e−2	−5.4729e−1

Table 5.4: Comparison of the UbiSOM, Online SOM and PLSOM algorithms across all data streams.

### Convergence and Exploratory Cluster Analysis

The first presented results target a numerical comparison between the UbiSOM algorithm against the *Online* SOM and the PLSOM algorithms in terms of quality metrics across selected artificial data streams. Due to the similar mapping performed by PLSOM and DSOM, the later was left out of this numerical comparison. Table 5.4 summarizes the obtained values for the previously used measures in the UbiSOM PSA (Section 5.5.3), namely the Mean  $E'_q(t)$ , Mean  $\lambda(t)$  and Mean topographic error ( $\overline{TE}$ ), for all algorithms. Also, the optimization function of Eq. (5.12) was applied to the individual results.

Results indicate that the UbiSOM algorithm obtains the best compromise results between all measures, for these data streams. The PLSOM exhibits a lower mean  $E'_q(t)$  on all data streams, except *Gauss*, because it does not map the input space density, as previously illustrated, being able to quantize the outer region of the Gaussian cloud. While this may turn out to be useful in some specific application, it does not bring any positive impact in terms of exploratory cluster analysis. In terms of mean topographic error, the PLSOM is consistently worst across all data streams, whereas the classical Online SOM usually attains the best results. Not surprisingly, the Online SOM exhibits higher mean  $E'_q(t)$  (worst) values because its convergence is slower, due to the annealing process throughout the entire data stream. In the “harder” data stream, i.e., the *Clouds* data stream, the UbiSOM algorithm obtained best values for the mean  $\lambda(t)$  and mean TE, with a very close value in respect to the (best) PLSOM mean  $E'_q(t)$  result. However, this metric implies nothing regarding actual convergence of topological ordered maps to distributions, i.e., it must be taken into account with the other two measures.

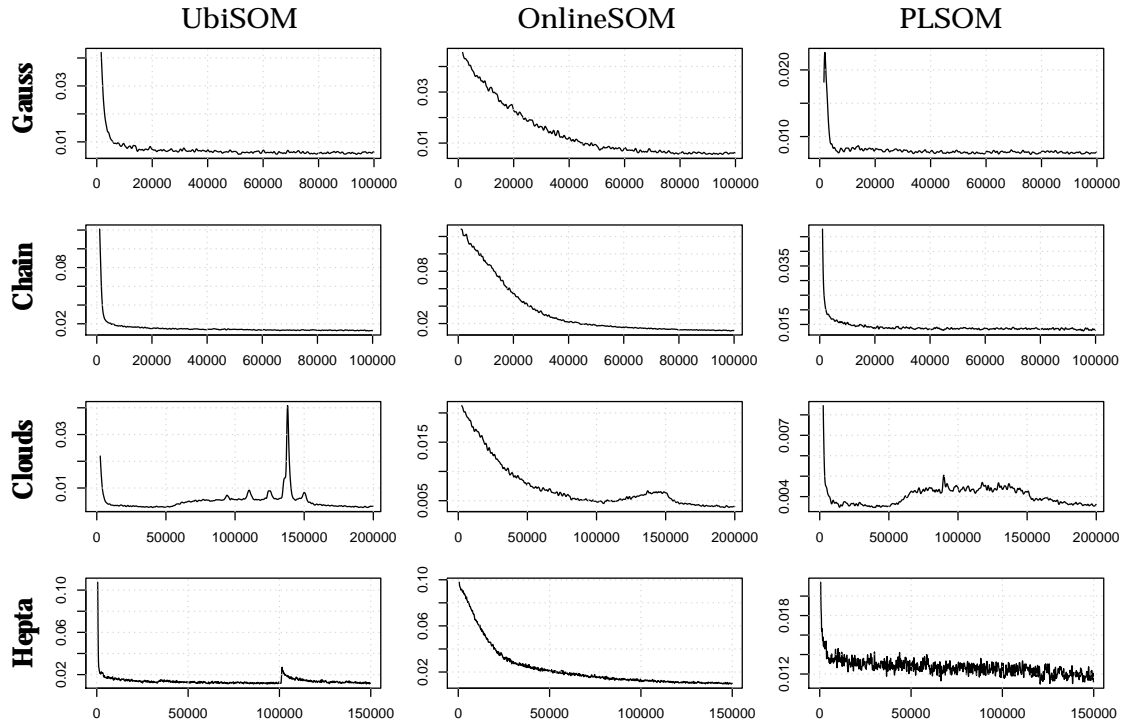


Figure 5.15: Average quantization error  $\bar{q}e(t)$  of tested SOM variants across all data streams. Results for the UbiSOM, Online SOM and PLSOM in left, center and right columns, respectively. The rows regard the *Gauss*, *Chain*, *Hepta* and *Clouds* data streams, respectively.

Figure 5.15 compares the convergence trend of the three algorithms in terms of the average quantization error  $\bar{q}e(t)$ . As with the UbiSOM, this should highlight how each algorithm performs over the data streams regarding the quantization procedure. The  $\bar{q}e$  values for the classical Online SOM and PLSOM are obtained as in UbiSOM (described in Section 5.3.3), using the  $T$  parameter of the UbiSOM for the respective data stream. This is considered fair for all algorithms, since it is simply evaluating the trend of the quantization error for each algorithm.

In general, it can be seen that the UbiSOM and PLSOM algorithms converge rapidly to the initial distributions. PLSOM achieves slightly lower values, consistent with Table 5.4. As expected, the convergence of the Online SOM is dictated by the monotonic decrease of the learning parameters. In respect to the *Clouds* data stream, the  $\bar{q}e(t)$  trend at the end of the data stream seen to indicate that all algorithms, in some way, were able to quantize the final observations. Otherwise, the metric would not show a decreasing trend. However, this gives no indication of proper convergence, i.e., “forgetting” old observations. By Table 5.4 it can be seen that the Online SOM and PLSOM exhibit a mean neuron activity of 89% and 94%, respectively, compared to 98% of the UbiSOM. This gives an indication that the UbiSOM performed better along this data stream. Finally, concerning the *Hepta* data stream, no indication that the *Online* SOM was able to react to the cluster

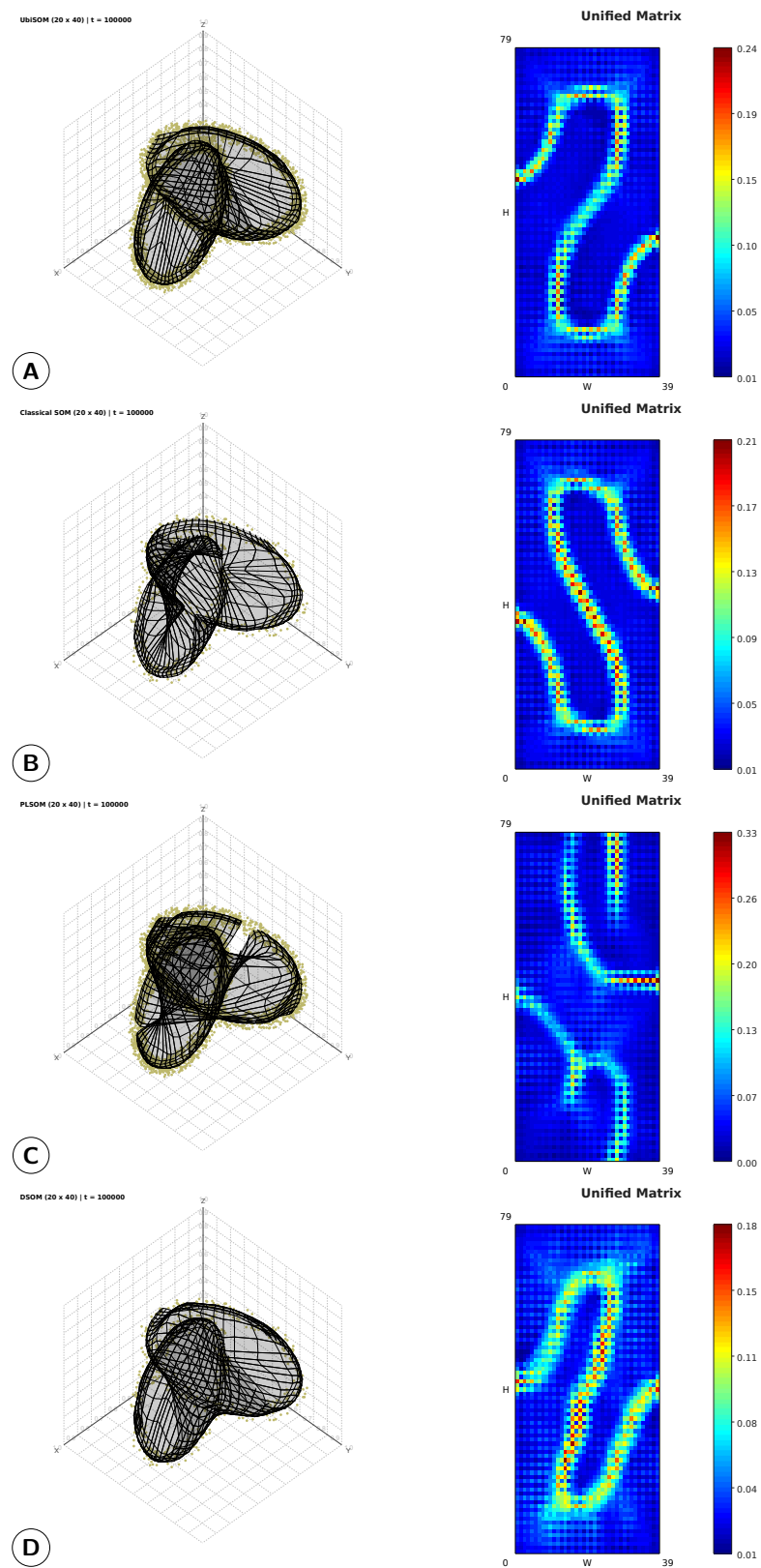


Figure 5.16: Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PL-SOM at the end of the stationary *Chain* data stream.



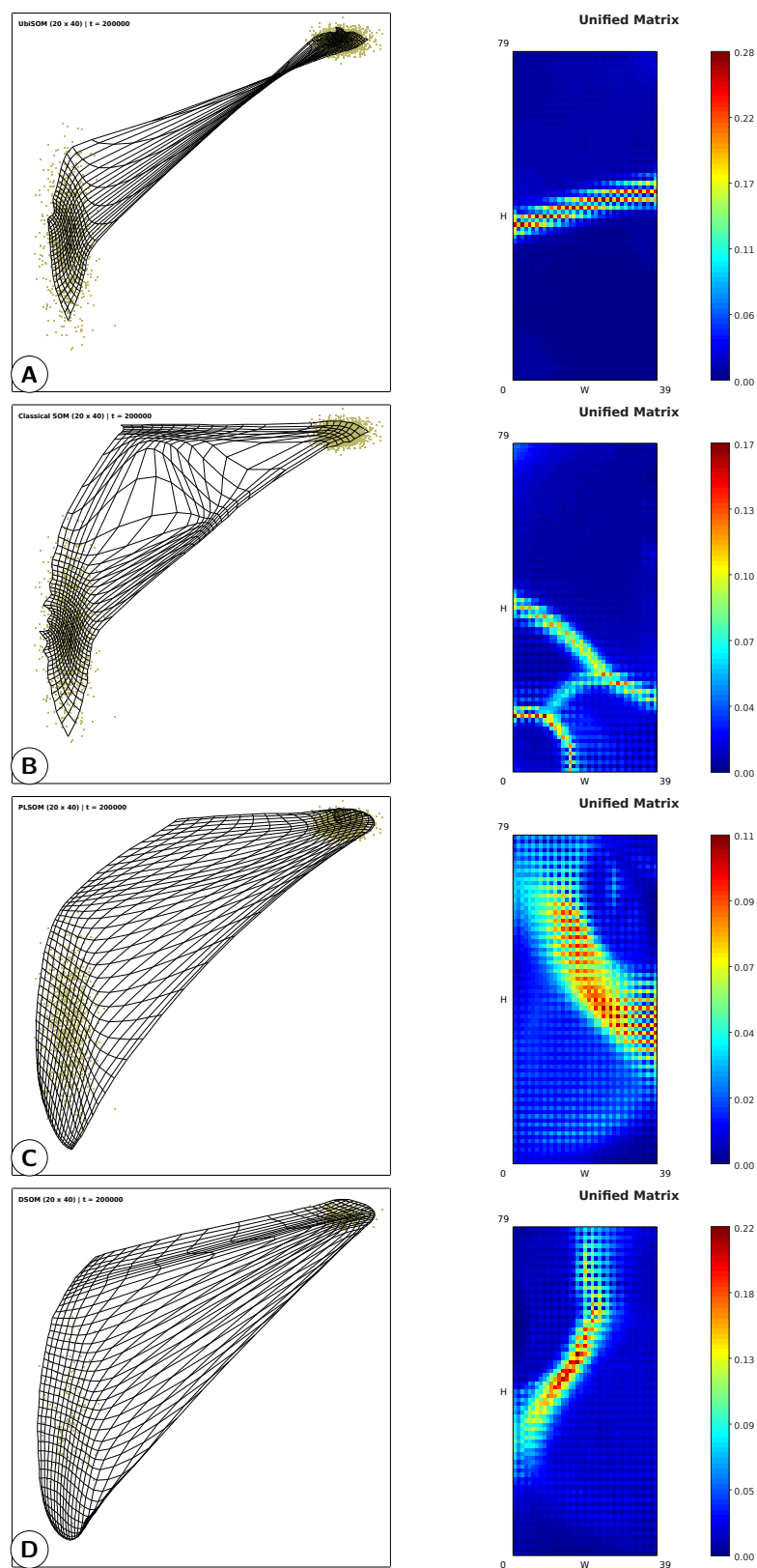


Figure 5.17: Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PL-SOM at the end of the non-stationary *Clouds* data stream.



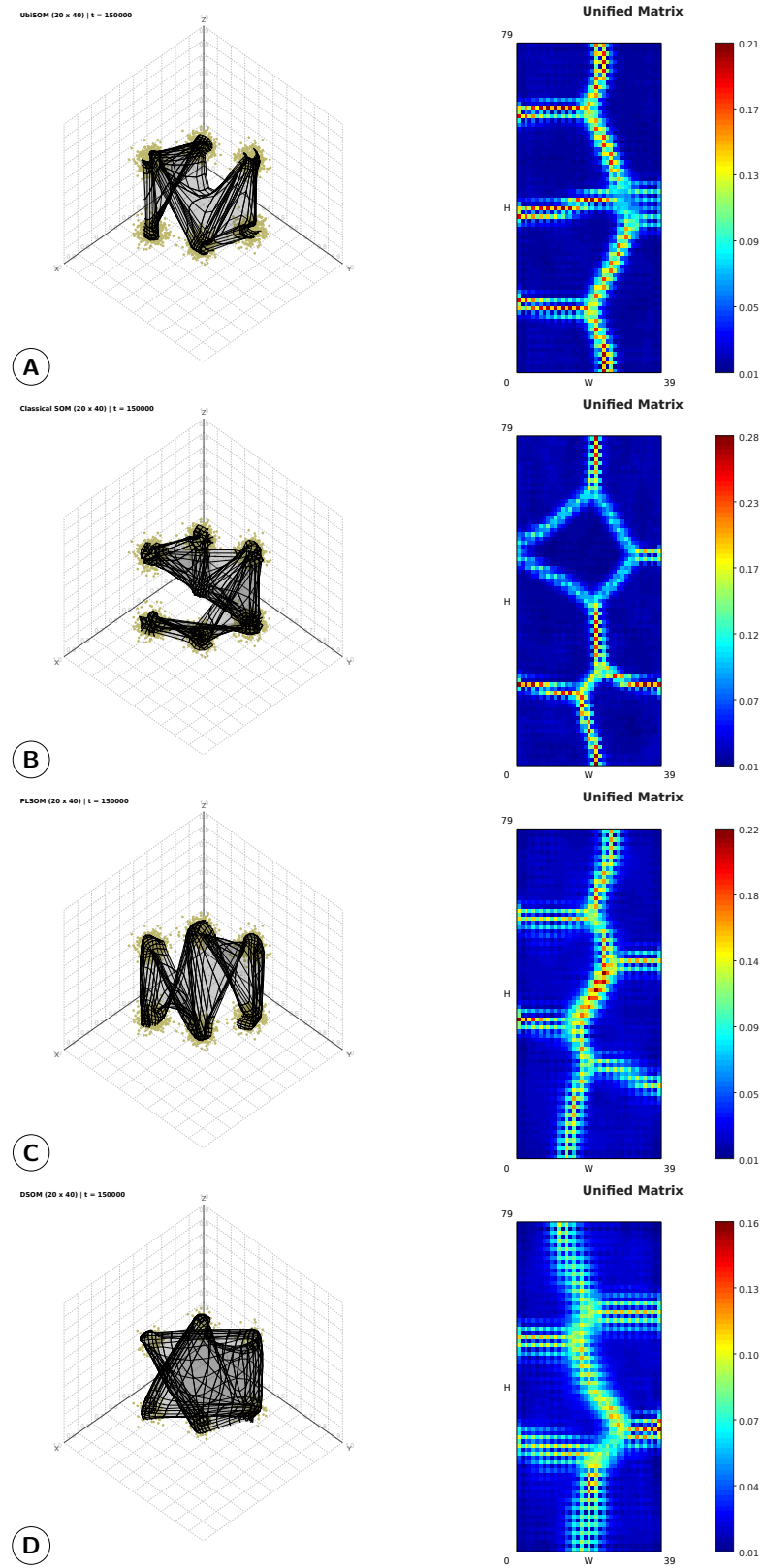


Figure 5.18: Comparison of final maps and U-Matrices of UbiSOM, OnlineSOM and PL-SOM at the end of the non-stationary *Hepta* data stream.

disappearance is present. The  $\overline{qe}(t)$  trend of the PLSOM algorithm convergence suggests that it does not achieve a stable convergence to this distribution.

In order to support the above inferences, to visualize the actual convergence of the maps and compare results for exploratory cluster analysis, Figures 5.16, 5.17 and 5.18 compare final maps and U-matrices obtained for the *Chain*, *Clouds* and *Hepta* data streams, respectively:

**Chain** Regarding this data stream, both the UbiSOM and *Online* SOM give identical results, the difference is only in the orientation of the map; both U-Matrices identify the two clusters clearly. The PLSOM final lattice contains topological defects that affect the exploratory cluster analysis. This is consistent with the higher mean topographic error of PLSOM in Table 5.4. DSOM is able to learn the distribution, but the U-Matrix is not as “sharp” as the UbiSOM or *Online* SOM, due to the density matching issues.

**Clouds** Compared to the UbiSOM results, obtained lattices for the *Online* SOM, PLSOM and DSOM show that these algorithms were not able to correctly adjust to the non-stationary distributions. This data stream clearly shows the deficiencies of these algorithms when dealing with non-stationary data, i.e., the *Online* SOM does not possess sufficiently high learning parameter values at the second half of the data stream to cope with the changes; the PLSOM algorithm uses an estimation of the input space diameter to compute the learning parameters, and; the DSOM only uses “local” error information to adjust learning parameters. While, e.g., this allows the PLSOM to achieve lower quantization errors (note that it covers the cluster clouds almost totally), it does not allow it to properly map the input space density and forget old observations. DSOM gives a similar result to PLSOM, but with a clearer U-Matrix.

**Hepta** This data stream also highlights the fact that the *Online* SOM, by decreasing the learning parameters with an annealing scheme, cannot react to the disappearance of a cluster latter in the data stream, when learning parameters attained very small values. Both PLSOM and DSOM, however, copped rather satisfactorily with this change. Nonetheless, the density matching issues of these proposals can be seen in the final lattice.

Hence, we can conclude that the UbiSOM algorithm is able to keep the original properties of the original SOM intact, while being able to cope with non-stationary data.

## 5.6 Feature Clustering Methodology and Evaluation

This section presents a feature clustering methodology for general SOM models, i.e., it is applicable either to Batch SOM models obtained in the StreamART2A/SOM methodology (see Chapter 4) as well as UbiSOM models. However, the evaluation in this section regards only the feature clustering capabilities of the UbiSOM along a data stream. Later in Chapter 7 the same procedure is applied to StreamART2A/SOM models in a real-world application. Feature clustering in a stream setting can be associated with time series clustering, given the goal is to produce a partition of the time series into groups, as discussed in Section 2.6.2.

### 5.6.1 Methodology

As presented in Section 2.4.5, the component planes visualizations allow the detection of correlated features, a process that is sometimes referred to as correlation hunting in SOM literature (Vesanto and Ahola 1999). The proposed methodology consists in extracting UbiSOM models along the data stream and generate the clustering of the component planes, and consequently the features, using an agglomerative hierarchical procedure. In (Silva and Marques 2010a) the methodology was proposed using a distance matrix between the component planes constructed by the inner product between matrices. Let  $X$  and  $Y$  be two normalized component planes, i.e.,  $X$  and  $Y$  are  $[width \times height]$  matrices with  $x_{\in X}, y_{\in Y} \in [0, 1]$ . Equation 5.13 formalizes a correlation function  $r_{in} : X \times Y \rightarrow [0, 1]$ , which is based on the inner product between matrices, satisfying the following conditions, for all non-zero scalars  $a$  and  $b$  and for  $X$  and  $Y$  not both zero:

$$\begin{aligned}
 C1: \quad & r_{in}(aX, Y) = r_{in}(X, bY) = r_{in}(X, Y) \\
 C2: \quad & r_{in}(X, Y) = r_{in}(Y, X) \\
 C3: \quad & r_{in}(X, Y) = 1 \quad \text{if } X = bY \\
 C4: \quad & r_{in}(X, Y) = 0 \quad \text{iff } X^T Y = 0
 \end{aligned}$$

$$r_{in} = \frac{tr(X^T Y)}{\sqrt{tr(X^T X) tr(Y^T Y)}} \quad (5.13)$$

where  $tr(\cdot)$  is the *trace* operation, i.e., the sum of the diagonal elements.

For the hierarchical procedure a distance matrix containing the distances between all component planes is generated. The distance between component planes  $X$  and  $Y$  is given by:

$$D(X, Y) = 1 - r_{in}(X, Y). \quad (5.14)$$

The distance matrix is then clustered by an agglomerative hierarchical procedure with complete-linkage (see Section 2.4.7.2). At each step the procedure recursively merges the

two most closest clusters (of component planes), i.e., the two clusters separated by the shortest distance are combined. The definition of “shortest distance” is what differentiates between the different agglomerative clustering methods. In complete-linkage clustering the link between two clusters contains all element pairs and the distance between clusters equals the distance between those two elements (one in each cluster) that are farthest away from each other. The algorithm stops when all features have been grouped into a single cluster and provides a *dendrogram* representing the obtained clustering.

In other works, to organize the presentation of component planes in a batch setting, e.g., (Pérez-Urbe 2007), the distance between component planes is given by the Person’s correlation coefficient, which is defined between  $[-1, 1]$ . Effectively, the Pearson’s correlation coefficient  $r_{pearson}(X, Y)$  can be computed as in Eq. (5.13), but with the matrices (component planes) normalized around their respective means. From here, two things are relevant: *i.*) it assumes the data is normally distributed, and; *ii.*) inversely correlated component planes are also clustered together. With the Pearson’s correlation, the distance between component planes should be obtained by  $D(X, Y) = 1 - \text{abs}(r_{pearson}(X, Y))$  to obtain positive distances.

### 5.6.2 Evaluation

The goal is to evaluate the proposed methodology against an artificial data stream containing known correlations between features, i.e., the time series. Differences between the usage of the inner product versus the Pearson’s correlation as distance metrics are also of interest.

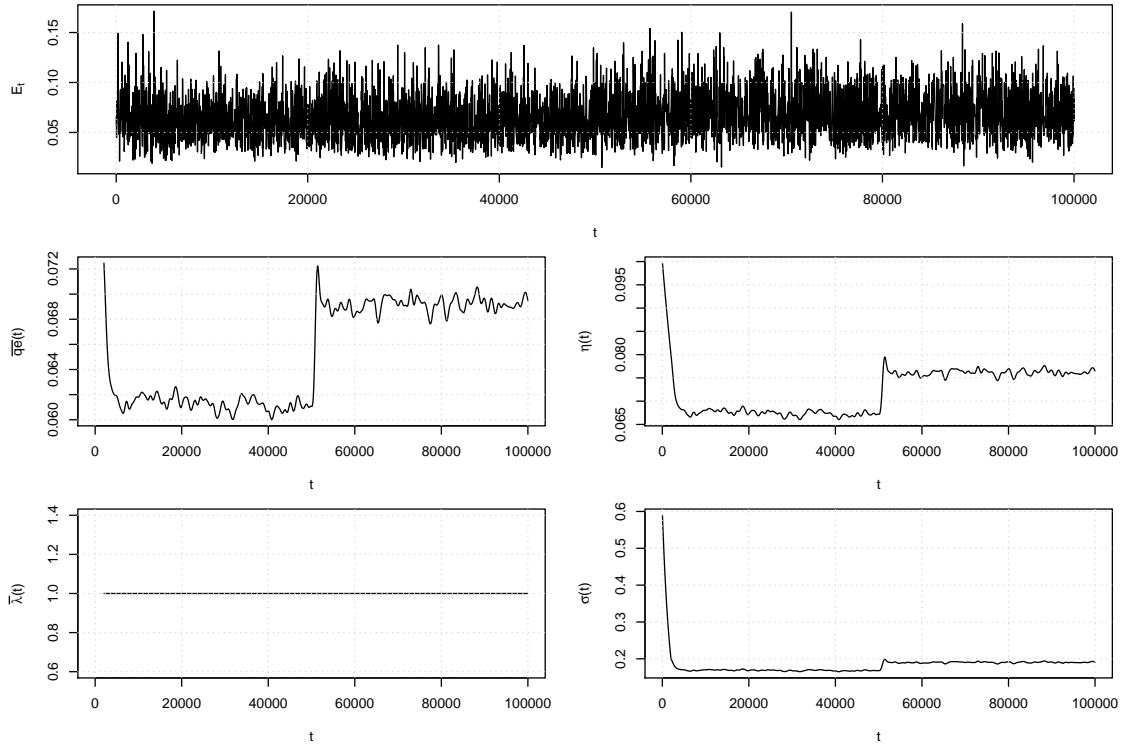
### 5.6.3 Data and Parameterization

The methodology evaluation was performed using the *Correlated* data stream. This data stream consists of 100 000 observations and 5 pairs of correlated time-series (hence,  $d = 10$ ) with different degrees of correlation. Table 5.5 summarizes the correlations between the individual features in both halves of the data stream. As before, the data streams was normalized in the unit hypercube, i.e.,  $\mathbf{x}(t) \in [0, 1]^d$ . The parameterization of the UbiSOM algorithm remained as in the previous sections, but with  $T = 2000$  and  $\beta = 0.8$ , as suggested at the end of Section 5.5.3.

### 5.6.4 Results

UbiSOM models were extracted at  $t = 30\,000$  and  $t = 80\,000$ , i.e., approximately at the middle of each stationary phase of the data stream. The learning procedure throughout the data stream is illustrated in Figure 5.19. If we look at the evolution of the learning parameters (determined by  $d(t)$ ), we can see that the algorithm reacts to an abrupt change at  $t = 50\,000$ , without the need to perform a *reset*, i.e., change back to the ordering state. After the algorithm converges to the new distribution it does not attain the same average

Degree of Correlation	First half, $t \in [0, 50\,000]$	Second half, $t = [50\,001, 100\,000]$
0	{a,b}	{g,h}
0.4	{c,d}	{c,d}
0.8	{e,f}	{e,f}
1	{g,h}	{i,j}
-0.6	{i,j}	{a,b}

Table 5.5: Underlying correlations in the artificial *Correlated* data stream.Figure 5.19: Learning evolution of UbiSOM over the *Correlated* data stream.

quantization error as during the first half. This can be explained by the new underlying distribution, where the time series are generated from *Gaussian* distributions with higher variances. The  $\bar{\lambda}(t)$  is always one because this data stream effectively describes a multi-dimensional *Gaussian* cloud, i.e., similar to Figure 5.14, but in a higher dimensional space. Overall, the learning parameters do not attain stable values because this data stream is inherently non-stationary, besides the change point. This is because the pairs of time series that are least or not correlated indeed contribute to this mild non-stationary characteristic.

Figure 5.20 depicts the obtained component planes and resulting hierarchical clustering results for the UbiSOM model at  $t = 30\,000$ . It can be seen that both distances

functions, i.e., that use the inner product  $r_{in}(X, Y)$  and Pearson's correlation coefficient  $r_{pearson}(X, Y)$ , are able to sequentially cluster the more correlated time series according to their pair-wise similarity, e.g.,  $\{g, h\}$ ,  $\{e, f\}$  and  $\{c, d\}$ , which have correlations of 1, 0.8, and 0.4, respectively. As expected, the Pearson's coefficient clustered the pair  $\{i, j\}$  which has a correlation of  $-0.6$ , whereas the inner product found this pair to be very dissimilar. Both distances found the pair  $\{a, b\}$  to be very dissimilar, which is sound because they were generated with no correlation. However, both clustered together  $\{\{c, d\}, a\}$  and  $\{\{g, h\}, b\}$ , hence indicating that there is some degree of "coincidental" correlation between these time series. One difference that clearly emerges regards the final steps of the agglomerative process: while the Pearson's correlation finds the top clusters very dissimilar, i.e.,  $D(X, Y) \approx 1$ , the inner product clusters them at different levels. This is a consequence of the Pearson's correlation assuming normally distributed data.

Similarly, Figure 5.21 contains results for the model at  $t = 80\,000$ , after the change in the pair-wise correlations. Again, the methodology is able to indicate the most similar time series, e.g.,  $\{i, j\}$ ,  $\{e, f\}$  and  $\{c, d\}$ , with underlying correlations of 1, 0.8 and 0.4, respectively; the pair  $\{a, b\}$  is also found to be similar by the Pearson's correlation with an underlying correlation of  $-0.6$ . Both also indicate a coincidental correlation between  $\{\{i, j\}, g\}$ .

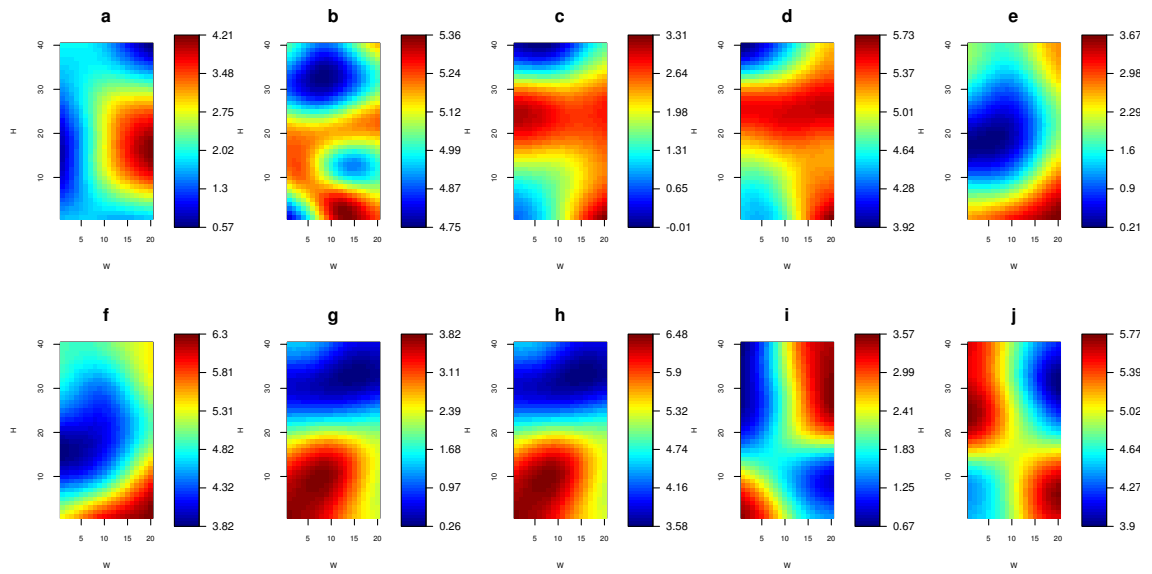
The above results seem to confirm the ability of the UbiSOM algorithm in also performing feature clustering along a data stream, enabling it as a versatile cluster analysis method for data streams. This characteristic is not found in other available clustering methods and emerges from the topological ordered abstraction of the UbiSOM codebook along non-stationary data. Whether some application should use the inner product or Pearson's correlation is highly dependent on the goal. For example, when dealing with financial data for the purpose of portfolio selection, the goal is to find clusters of similar stocks and then diversify the choice of stocks among these groups. In this particular case the Pearson's correlation is not good because it will cluster together inversely correlated stocks, which is ultimately against the proposed goal. Also, the inner product seems to better differentiate the different levels of similarity between the component planes.

As a final note, features can be grouped in  $k < d$  clusters by a *cuttree* algorithm applied over the *dendrogram*. This is exemplified later in a real-world application in Chapter 7.

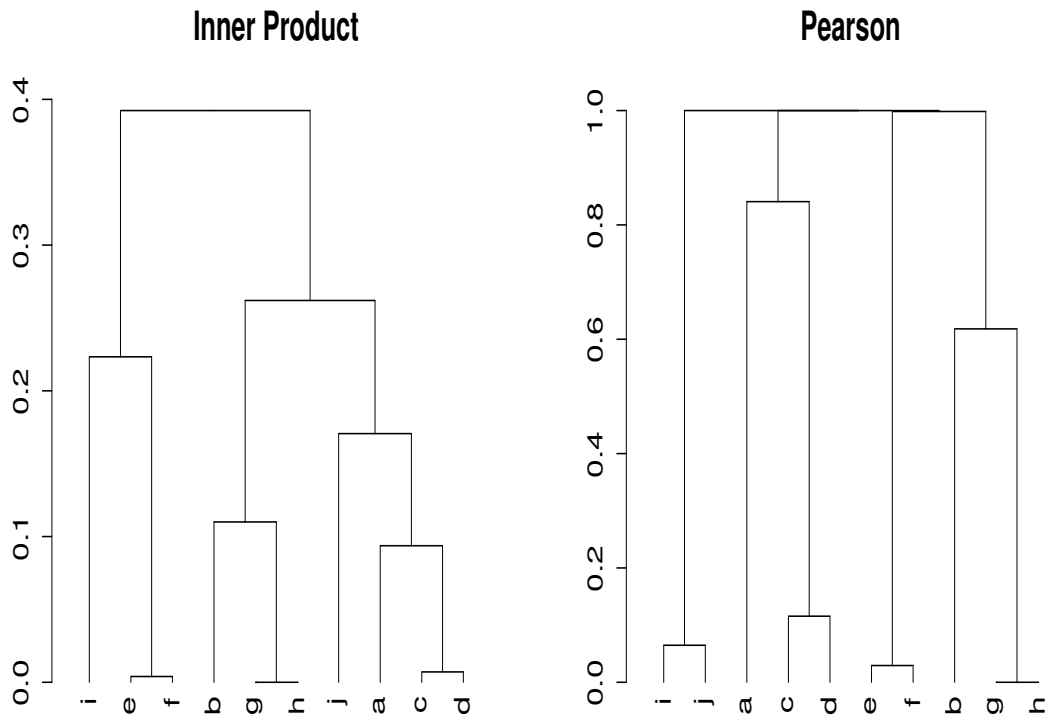
## 5.7 Assessment Regarding Established Requirements

In respect to the requirements put forward in Section 2.5.1 for SOM variants dealing to data streams, they are addressed by UbiSOM as follows:

**Fixed topology.** UbiSOM uses a regular lattice as the original SOM. No units are added or removed throughout the learning process. This allows the applicability of standard

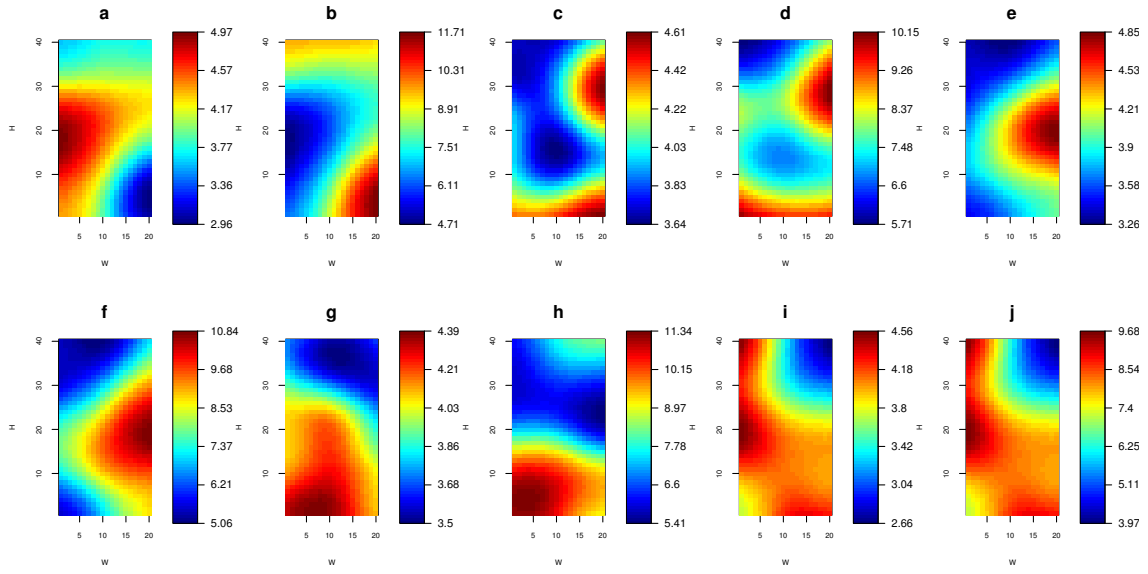


(a) Component Planes.

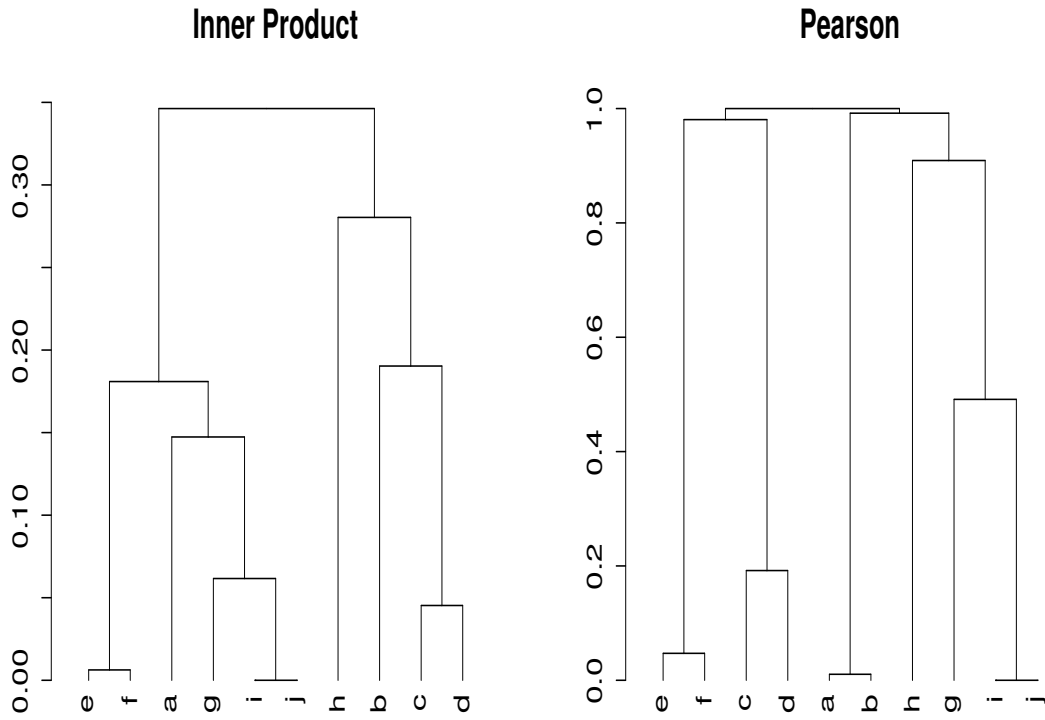


(b) Hierarchical clustering obtained using  $r_{in}(X, Y)$ . (c) Hierarchical clustering obtained using  $r_{pearson}(X, Y)$ .

Figure 5.20: Feature clustering from UbiSOM model at  $t = 30\,000$  when learning the *Correlated* data stream.



(a) Component Planes.



(b) Hierarchical clustering obtained from  $r_{in}(X, Y)$ . (c) Hierarchical clustering obtained from  $r_{pearson}(X, Y)$ .

Figure 5.21: Feature clustering from UbiSOM model at  $t = 80\,000$  when learning the *Correlated* data stream.



visualization techniques, e.g., U-Matrix and component planes;

**Time-independent learning parameters.** Although during the ordering state an annealing scheme continues to be used, it is only during a comparatively short period of time to allow the unfolding of the map and to obtain the first values of the global assessment metrics. From this point forward, learning parameters are estimated independently of the iteration number;

**Proper convergence.** UbiSOM replicates the results of the original SOM algorithm during stationary states of the data stream, hence achieving an approximated VQ of the input space. This ability is not found in PLSOM and DSOM, for example. This is a consequence of the estimation of learning parameters from the proposed assessment metrics and is one of the major contributions of this research;

**Incremental processing of observations.** Assuming a sufficiently high number of representative observations characterizing a stable state of the distribution, the UbiSOM processes one observation at a time and does not store any past observations;

**Compactness of the model.** This property is assured by the fixed topology and bounded by the selected lattice size;

**Dealing with evolutionary data.** *Plasticity* in the UbiSOM is guaranteed by the behavior of the assessment metrics, increasing learning parameters when change is signaled;

**Robustness to noise.** This is the aspect that needs further study. Although the triple-cascade moving average, and moving averages in general, introduce some robustness to noise, the presented evaluation cannot establish with certain this property. This is an aspect that should be addressed in future work;

**Handling high-dimensional data.** As the original properties of the SOM are maintained, we can assume that the UbiSOM continues to be regarded as a projection algorithm of high-dimensional data to a lower dimensional space.

Based in this analysis, we can conclude the UbiSOM fulfills all the established requirements, although with some reservations regarding *robustness to noise*.

## 5.8 Remarks and Future Work

In this chapter the Ubiquitous Self-Organizing Map (UbiSOM) was presented as a novel SOM variant tailored for non-stationary data streams. The UbiSOM is able to properly converge in the sense of a VQ procedure during stationary phases of the data stream,

while being able to react to changes in the underlying data stream. This is achieved by estimating learning parameters through proposed global assessment metrics, namely the *average quantization error* ( $\overline{qe}(t)$ ) and *average neuron utility* ( $\overline{\lambda}(t)$ ), which are moving averages over computed values at each iteration. The former metric indicates the trend of convergence to the underlying distribution, i.e., as the lattice converges the  $\overline{qe}(t)$  attains increasingly lower values; on the other hand, if the underlying distribution drifts from the quantization of the current UbiSOM codebook, it increases. While the  $\overline{qe}(t)$  can be considered a baseline assessment, it may not react to some exemplified changes, e.g., disappearance of clusters. Hence, the later metric assesses the percentage of neurons actively being updated. If it decreases, then some regions of lattice have become obsolete, i.e., are not quantizing current observations, hence signaling change. Weighted by  $\beta \in [0, 1]$  in the proposed *drift function*, they are used to estimate the learning parameters throughout the data stream. Regarding the assessment metrics, a *triple-cascade moving average* was proposed to allow the  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$  functions to be differentiable along time and ultimately provide a smooth increase and/or decrease of learning parameters. Another important aspect is the UbiSOM two-state algorithm, providing an *ordering state* where the initial unfolding of the map takes place, and; the *learning state*, as the main state of the algorithm. In case of such a change in the underlying distribution to which the UbiSOM cannot cope with, the algorithm transitions back to the ordering state.

This proposal contrasts with the two-stage approach presented in Chapter 4, because the goal is to continuously achieve a SOM abstraction of the data stream's underlying distribution, instead of generating a SOM model offline. While each approach have their strengths and weaknesses (which are compared in Chapter 8), the main strength of the UbiSOM is that it enables real-time knowledge discovery along a data stream through its visualization procedures. On the other hand, the correct parameterization of the UbiSOM for a particular problem is harder than in the StreamART2A/SOM methodology.

Although the evaluation of the algorithm was performed solely over artificial data streams, in Chapter 7 real-world applications of the UbiSOM are presented, leveraging its abilities in different scenarios of exploratory cluster analysis, both involving traditional observation clustering and feature clustering. Those applications should further motivate the use of the UbiSOM in real-world scenarios.

Several issues remain and should be addressed in future work, namely:

**Noisy data and additional evaluations.** The impact of noise in the models should be further studied, as it is one of the established requirements partially fulfilled (see Section 5.7). Also, while in theory and through several experimental results not included in this manuscript, the choice of the UbiSOM parameters is independent of the lattice size, this should be formally evaluated;

**Algorithm improvements.** Study the possible decoupling of the ordering state duration and global assessment metrics window size; at the moment both are determined by the parameter  $T$ . While this may be of interest, e.g., imposing a short, medium or long-term trend in the assessment metrics independently of the ordering state duration, the window size cannot be greater than the ordering duration, since we still have to wait for the first values of  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$  to be available. Also, the possibility of dynamic adjustment of the parameter  $\beta$  is interesting, i.e., increasing during stable states of the distribution, whereas decreasing during non-stationary phases. While this could potentially eliminate the need to parameterize  $\beta$ , it remains to be studied how to achieve this.

**Automatic change detection.** Inclusion of a change detection mechanism, either over values of the drift function  $d(t)$  or with an independent procedure. Such methods were already discussed in Section 4.6, regarding automatic change detection in the StreamART2A algorithm. In the UbiSOM, model assessment is an intrinsic mechanism of the algorithm, rather than a separate procedure as in the StreamART2A, making it more attractive to develop change detection procedures over  $d(t)$  values. This is supported by the fact that the drift function  $d(t)$  is consistent regarding the increase of its values in the presence of change, as opposed to only the  $\overline{qe}(t)$  values, which may not (please recall Section 5.3.3). Opting for independent procedures such as in (Kifer, Ben-David, and Gehrke 2004) and (Ho and Wechsler 2007) would still impose additional computational complexity.

The added benefit would still be the same as discussed previously, i.e., to allow storage of UbiSOM codebook snapshots for later analysis when change is detected. Consequently, insight on what changes have occurred can be obtained by comparing consecutive snapshots.

**Confidence in the model.** Finally, to address the confidence level in the UbiSOM model at any time. While during the ordering state no cluster analysis should be performed, there is currently no established way, e.g., a single value, to indicate the level of approximation of the model; only the drift function  $d(t)$  trend.

To this extent, but not sufficient, in Chapter 7 two new visualizations for the UbiSOM are presented, leveraging the extended information within the UbiSOM neurons, e.g.,  $t_k^{bmu}$  and  $t_k^{update}$ . From these, the user can get a grasp on whether the entire map is being used to quantize the current underlying distribution – this can be regarded as a visual alternative to a simple numerical value.



# 6

## Distributed and Collaborative Learning of Ubiquitous Data Streams

Research is what I'm doing when I don't know what I'm doing.

---

WERNHER VON BRAUN, GERMAN SCIENTIST (1912-1977)

The previous chapter introduced the *Ubiquitous Self-Organizing Map* algorithm, capable of learning multi-dimensional non-stationary data streams. This enables continuous summarization of an incoming data stream and real-time exploratory cluster analysis. This chapter extends the use of the UbiSOM to ubiquitous environments, where data is inherently distributed, proposing methodologies to leverage local and global models. The name “*ubiquitous*” was deliberately chosen with these additional applications to ubiquitous environments in mind.

### 6.1 Chapter Overview

This chapter presents a set of methodologies to exploit the UbiSOM in distributed and collaborative learning settings, inspired by existing proposals that focus in the trade-off between local and global models. The chapter is organized as follows: in the next section, motivation and aim are presented, illustrating a general motivating scenario. In Section 6.3 the proposed methodologies for distributed and collaborative learning strategies are described. Validation and evaluation of the proposed methodologies are made in Section 6.4, using artificial data streams. Finally, Section 6.5 concludes the chapter, including foreseen future work.



Figure 6.1: Motivating example for distributed and/or collaborative learning strategies, focusing on the generation of local and global models.

## 6.2 Motivation and Aim

In a truly ubiquitous environment data mining is inherently distributed. Consider the illustrative scenario portrayed in Figure 6.1. It depicts a network of sensors in a city, where each sensor collects some measurements, e.g., temperature, humidity, luminance, which are the features of the gathered observations; the set of these observations is a *multidimensional* data stream. A traditional approach would consist in a centralized process that would gather data from all sensors and then apply a data mining algorithm, e.g., obtain a SOM model. However, this raises two problems: first, the amount of data that each node needs to communicate is restrained by the available bandwidth; second, and consequently, as the number of sensors grow, the scalability of the approach degrades.

As these sensors advance technologically with increasing computing, storage and communication capabilities, the entire process can be distributed and parallelized throughout the entire network of processing nodes, leveraging a two-level approach where each sensor can separately process its own data and then a centralized process combines all the individual results (models), transmitted by each sensor, to define a global model. In this scenario, it is assumed that the sensors are more than mere data-collectors. Rather, these sensors are full-fledged information processors capable of running a (computationally

efficient) data mining algorithm, as the UbiSOM.

Given that an UbiSOM model can be seen as a compact representation of the current underlying distribution of a local data stream (topological ordering aside), its prototypes are coarser representations of the learned observations which can be used as inputs to a global SOM model. Regarding a purely centralized process, this can greatly reduce the amount of information each sensor has to communicate, by transmitting at most only the representative prototypes of data.

Consequently, the aim of this chapter is to propose and validate methodologies that allow UbiSOM codebooks, representing local models of a processing node, e.g., smart sensor, smartphone, to be transmitted to other nodes so as to generate global models. In the example portrayed in Figure 6.1 a centralized global model would allow knowledge discovery from the entire city's environment. This type of learning strategy will be called *distributed learning*. If the processing nodes are able to move, e.g., smartphones, then the interest is to allow those devices to share their models with other devices in their vicinity, as they come into proximity. This will be called *collaborative learning* and the motivating application regarding the portrayed example relates to *participatory sensing* (Burke et al. 2006; Dutta et al. 2009), characterized by the “*deployment of mobile devices to form interactive, participatory sensor networks that enable public and professional users to gather, analyze and share local knowledge*”. In this case a single mobile device can have its local model, depicting its subjective view of the environment, while maintaining a global model from “knowledge” obtained from other devices. These approaches are in line with current proposals leveraging local and global models presented in Section 2.6.3 as a means to address ubiquitous data streams.

## 6.3 Distributed and Collaborative Learning

### 6.3.1 Methodology Overview

The proposed methodology targets generation of *global* SOM models from distributed UbiSOM *local* models within an ubiquitous network of nodes, each processing a data stream pertaining the same *problem*/task. This relies on the premise that, at any given time, each UbiSOM codebook in a distributed location is a compact representation of the current underlying distribution of the processed data stream. As such, each codebook is a data stream summary that can be sent to a centralized location — *distributed learning*, or shared with other nodes — *collaborative learning*. The union of such codebooks can effectively be used to train another SOM model (Vesanto and Alhoniemi 2000).

Consequence of the topological ordering of the UbiSOM prototypes, not all are true representatives of data — please recall the “unrepresentative” prototypes (or dead-units) issue addressed in Section ???. To overcome this, which would affect the generation of

correct global models, the codebooks are “filtered” before being transmitted. This filtering is based on an aging mechanism where only prototypes that have been recently been chosen as BMUs are considered true representatives.

Distributed learning is based on the centralization of several distributed UbiSOM filtered codebooks that are then taken as a static dataset to train a global SOM model. In this proposal the Batch SOM algorithm is applied because there are known parallel implementations that can scale better with increasing distributed codebooks, e.g., (Silva and Marques 2007). Therefore, this procedure allows the generation of a global model describing the current collected data in a group of nodes.

Collaborative learning, on the other hand, targets nodes that move throughout the physical space and can share their models through opportunistic networks, e.g., proximity based peer-to-peer networks. This learning strategy implies that each node contains two UbiSOM models: the *local model*, responsible for maintaining a subjective view of data acquired by the node, and; the *global model* that learns from prototypes acquired from the local model codebook together with prototypes obtained from other nodes. These prototypes are stored in a *global buffer*, whose contents vary as the nodes interact. In order to distinguish the importance of prototypes originating from local and global models (the later are considered more important, since they represent partial distributions learned by other nodes), a prototype weighting mechanism is employed: firstly, by attaching to each global model a *map weight* that aims at translating the importance of the model codebook, as the model learns from increasing sources of prototypes. Secondly, the global UbiSOM update rule is modified to cope with prototypes with different weights (similarly to the Batch SOM update rule proposed in Chapter 4 to learn from micro-categories). Ultimately, a node’s global model describes what it has learned autonomously and through knowledge obtained from other nodes (Silva and Marques 2010b).

In both learning strategies codebooks are effectively merged, either when centralizing them or when a pair is shared between nodes. The merge procedure involves removing “duplicate” prototypes from the merged set, which would influence the global quantization procedure of the global SOM model.

### 6.3.2 Notation

The methodology presented in this chapter extends the usage of the UbiSOM algorithm to a *distributed/collaborative learning* setting. As such, the notation for the UbiSOM algorithm was already formalized in Table 5.1. Here, only the notation relevant to the description of the current methodologies is presented in Table 6.1.

Let  $N$  be the number of processing nodes in the network. Each node  $i$  has at least a local UbiSOM model whose codebook is formalized as  $\mathcal{L}_i$ ; for *collaborative learning* each node  $i$  additionally has a global UbiSOM model, whose codebook is referred to as  $\mathcal{G}_i$ , where  $i = 1, \dots, N$ . Local models learn directly from a data stream, while the global UbiSOM models learn from weighted prototypes stored locally in a *global buffer*. Different



Notation	Description
$k$	neuron <i>index</i>
$c$	index of <i>BMU</i>
$\mathcal{W}_k$	UbiSOM “extended” <i>neuron</i> : a tuple $\langle \mathbf{w}_k, t_k^{update}, t_k^{bmu} \rangle$
$t_k^{bmu}$	<i>timestamp</i> of last selection of $\mathbf{w}_k$ as the <i>BMU</i>
$\mathbf{x}(t)$	<i>observation</i> presented at time $t$ , where $\mathbf{x}(t) \in \mathbb{R}^d$
$N$	number of nodes in a network
$\phi$	weight of an UbiSOM model
$\mathcal{L}_i$	local UbiSOM codebook (set of prototypes) of node $i$ , where $\phi = 1$
$\mathcal{G}_i$	global UbiSOM codebook of node $i$ , where $\phi \geq 1$
$\mathcal{L}'_i$ and $\mathcal{G}'_i$	filtered codebooks
$\mathcal{N}_i$	prototypes shared by a node in <i>collaborative learning</i>
$\mathcal{D}$	a static dataset formed by UbiSOM prototypes
$T$	size of sliding window / length of ordering state (parameter)
$r_w$	<i>perceptive field</i> radius threshold used to detect overlap (parameter)

Table 6.1: Notation used in distributed and collaborative learning methodologies.

weights are attributed to prototypes according to their provenance, i.e., local models always have a weight  $\phi = 1$ , while the weight of global models increase as the respective global buffer is filled with prototypes from different nodes.

In either learning strategy (*distributed* or *collaborative*) a key aspect to consider is the selection of the prototypes to be transmitted to a central location or to other peers. Given an UbiSOM codebook abstracts the current underlying distribution of the data stream, the “unrepresentative” prototypes are not relevant to describe the underlying distribution. As such, prototypes to be transmitted are filtered according to their ability to actually represent data, i.e., only *best matching units* (BMU) are sent. As previously described in Chapter 5, each UbiSOM neuron  $\mathcal{W}_k$  is a tuple  $\langle \mathbf{w}_k, t_k^{update}, t_k^{bmu} \rangle$ , where  $t_k^{bmu}$  stores the last time neuron  $k$  was selected as the BMU for an observation  $\mathbf{x}(t)$ . By monitoring these *timestamps*, particularly in the learning state, one can get a grasp on the importance of individual prototypes in describing the current underlying distribution, effectively acting as an “aging” mechanism. Hence, only prototypes whose age does not exceed the parameter  $T$  are considered important in representing actual observations. Consequently, codebooks that go through the filtering process are formalized as  $\mathcal{L}'_i$  and  $\mathcal{G}'_i$ . Finally, when merging two codebooks, e.g.,  $merge(\mathcal{A}, \mathcal{B})$ , prototypes in  $\mathcal{B}$  that fall within the established perceptive field of radius  $r_w$  of any of the prototypes in  $\mathcal{A}$  are removed.

### 6.3.3 General Techniques

Both distributed and collaborative learning strategies involve common techniques, namely for filtering codebooks for transmission and merging codebooks from difference sources.

#### 6.3.3.1 Codebook Filtering for Transmission

UbiSOM codebooks are compact representations of learned data in the form of a set of prototypes. By transmitting only the codebooks, the amount of data that must be transferred is greatly reduced, as opposed to the entire learned data stream. However, in SOM models including UbiSOM, not all prototypes are truly representatives of learned data, i.e., the topological maps normally contain “unrepresentative” prototypes (see Section ??), which may act as boundary prototypes between true representatives, consequence of the neighborhood kernel. To avoid transmitting these prototypes, each UbiSOM neuron  $\mathcal{W}_k$  is expanded to include the time stamp  $t_k^{bmu}$  of the latest time the respective prototype was elected as the BMU (similarly to the *utility* time stamp), i.e., becoming  $\mathcal{W}_k = \langle \mathbf{w}_k, t_k^{util}, t_k^{bmu} \rangle$ . The idea is to filter the prototypes that are truly representing streaming observations.

A filtered UbiSOM codebook  $\mathcal{W}'$  is in fact the set of prototypes that were active in the last  $T$  iterations, as formalized in Eq. (6.1).

$$\mathcal{W}' = \left\{ \mathbf{w}_k : \{t - t_k^{bmu} \leq T\} \right\}. \quad (6.1)$$

This bears a similarity with the computation of the neuron utility measure (see Section ??), but instead allows to filter only the prototypes that were selected as BMU within the last  $T$  iterations of the algorithm.

#### 6.3.3.2 Merging Codebooks

In the proposed distributed/collaborative learning methodologies, sets of UbiSOM prototypes will serve as static datasets to generate global models. Moreover, these datasets will be composed by the union of several codebooks, where some may contain prototypes that overlap in the input space. Since keeping overlapping prototypes would create “artificially denser” regions, the merge procedure aims at removing prototypes that may occupy roughly the same input region as others.

Hence, the merge procedure consists in joining two codebooks of prototypes (from different models) in such a way that possible “duplicates” are removed. To formalize this procedure, let  $\mathcal{A}$  and  $\mathcal{B}$  be codebooks. In set-theory the *relative complement* of  $\mathcal{A}$  in  $\mathcal{B}$  is the set of elements in  $\mathcal{B}$ , but not in  $\mathcal{A}$ , and can be formalized as  $\mathcal{B} \setminus \mathcal{A} = \{\mathbf{w} \in \mathcal{B} : \mathbf{w} \notin \mathcal{A}\}$ . Consequently, the merge procedure is formalized in Eq. (6.2).

$$merge(\mathcal{A}, \mathcal{B}) = \mathcal{A} \cup (\mathcal{B} \setminus \mathcal{A}) \quad (6.2)$$

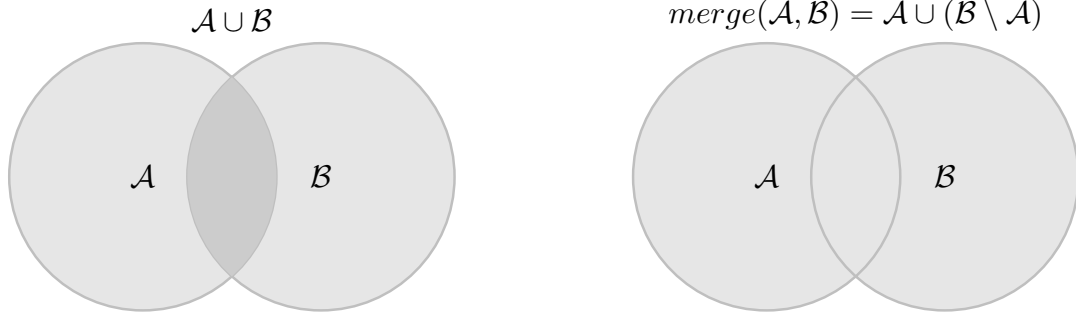


Figure 6.2: The merge procedure aims at removing duplicate prototypes from a pair of codebooks  $\mathcal{A}$  and  $\mathcal{B}$  that roughly overlap in the input space.

The rationale is that codebook  $\mathcal{A}$  is joined with prototypes of  $\mathcal{B}$  that are not in  $\mathcal{A}$ , hence resulting in a merge procedure without duplicates (Figure 6.2). Duplicate detection is inspired by the concept of the perceptive field of ART networks (see Chapter 4), i.e., by temporarily establishing very small perceptive fields around prototypes, two prototypes are considered duplicates if their perceptive fields overlap. The radius of the perceptive field is given by the parameter  $r_w$  and Eq. (6.3) establishes when two prototypes are considered duplicates.

$$\forall \mathbf{w}_i \in \mathcal{A}, \forall \mathbf{w}_j \in \mathcal{B} \quad isDuplicate(\mathbf{w}_i, \mathbf{w}_j) = \begin{cases} true & \text{if } \frac{\|\mathbf{w}_i - \mathbf{w}_j\|}{|\Omega|} < r_w \\ false & \text{otherwise} \end{cases} \quad (6.3)$$

Since the duplicate verification uses a normalized distance according to the dimensionality of the input space,  $r_w$  is dimensionless and should be chosen to a very small value, e.g., 0.01. This should effectively detect overlapping prototypes.

### 6.3.4 Distributed Learning

Figure 6.3 depicts a schematic of the *distributed learning* methodology. Assuming the existence of  $N$  distributed nodes, each running the UbiSOM algorithm over a separate data stream (pertaining the same problem, i.e., same features), the distributed learning procedure consists in centralizing all individual filtered codebooks of those models into a set of prototypes, to serve as a static dataset for another SOM model.

Let  $\mathcal{L}_1$  through  $\mathcal{L}_N$  be local UbiSOM models. The centralized dataset  $\mathcal{D}$ , formalized in Eq. (6.4), is obtained by the consecutive *merge* of the filtered models (as they arrive) and then used as input, i.e.,  $\mathbf{x}(t) \in \mathcal{D}$ , to a SOM algorithm to produce a centralized model. It should be noted that this dataset should be considered “stationary” and the order by which the input prototypes are fed into the algorithm should be random. This is a crucial aspect, otherwise the input prototypes would be presented by order of their

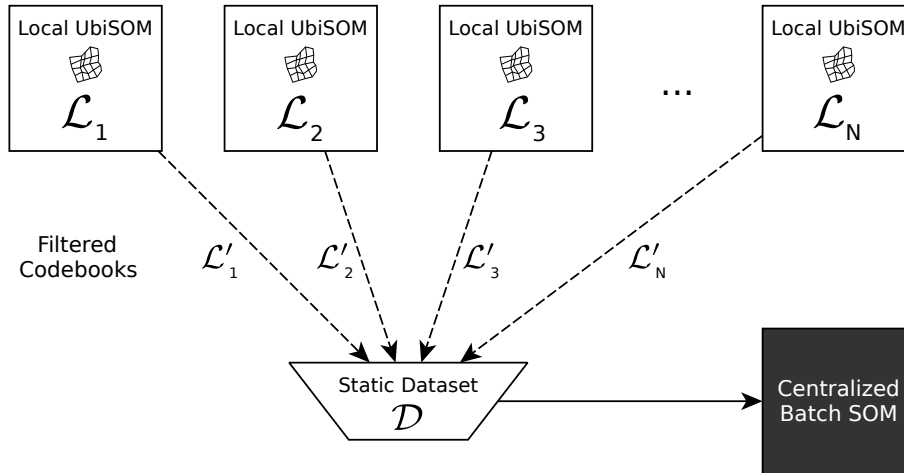


Figure 6.3: Distributed learning from distributed local UbiSOM models.

origin, explicitly describing  $N$  consecutive distributions.

$$\mathcal{D} = \text{merge}(\mathcal{L}'_1, \dots, \mathcal{L}'_N) \quad (6.4)$$

Therefore, one can, in theory, use any SOM algorithm over this dataset  $\mathcal{D}$  to produce the final model, ensuring that input prototypes are drawn randomly during presentation to the algorithm. In this proposed methodology, the *Batch* SOM algorithm is used for the centralized learning procedure, as in Chapter 4. Moreover, in extreme cases of a large number of distributed models and, consequently, a larger number of prototypes, prior to this research a parallel implementation of this variant (Silva and Marques 2007) was made available. This implementation scales in sub-linear time for increasing number of cores, due to the hybridization of data and lattice segmentation. Therefore, a parallel implementation of the Batch SOM addresses scalability issues that may arise with increasing distributed local models.

### 6.3.5 Collaborative Learning

The *collaborative learning* setting foresees each processing node possessing a local and global view of the problem/task. Collaborative learning is achieved among nodes by sharing their models with others. Hence, within each node, two models exist, as depicted in Figure 6.4 :

**Local model.** The local UbiSOM model processes the streaming data acquired by the processing node alone, generating a local model. It contains the *local codebook*  $\mathcal{L}_i$ .

**Global model.** The global UbiSOM model only learns from prototypes of data which can have different sources, i.e., from local and global codebooks. These prototypes are stored in the *global buffer* and recycled during presentation to the global UbiSOM; the contents of this buffer change throughout time. The global model can act simultaneously as a “long-term” memory for the local model, e.g., in case of

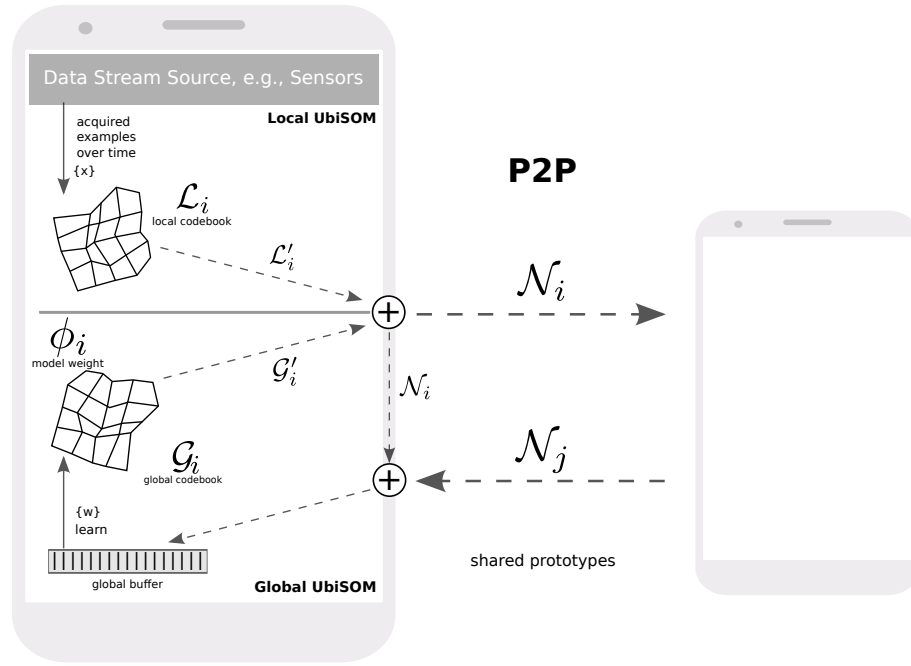


Figure 6.4: Collaborative learning framework.

changes in the underlying stream, and as a “collective” memory for models the device has interacted with, in a collaborative learning setting. The set of prototypes contained in this model is called the *global codebook*  $\mathcal{G}_i$ .

A weighting mechanism is introduced at the model level to establish a measure of the relative importance of prototypes from local and global models. Since the later may contain knowledge from other nodes, the prototypes are considered more important. Consequently, each UbiSOM model is extended with a weight  $\phi$ ; local models always have  $\phi = 1$  and the weight of a global model is incremented as the node interacts with other nodes, although being initialized with  $\phi = 1$ .

Figure 6.4 additionally illustrates the proposed methodology when two nodes interact in the ubiquitous environment, with the intent to collaborate. The methodology involves the following steps:

- Each node  $i$ :
  - i. Generates its local filtered codebook  $\mathcal{L}'_i$ ;
  - ii. Generates its global filtered codebook  $\mathcal{G}'_i$ , if the model already exists; otherwise  $\mathcal{G}'_i = \emptyset$ ;
  - iii. Merges the two codebooks into  $\mathcal{N}_i = \text{merge}(\mathcal{G}'_i, \mathcal{L}'_i)$ . Note that  $\mathcal{N}_i$  contains the prototypes from  $\mathcal{G}'_i$  plus the non-overlapping prototypes from  $\mathcal{L}'_i$ ;
  - iv. Sends  $\mathcal{N}_i$  to node  $j$ , receiving  $\mathcal{N}_j$  in return;
  - v. Merges  $\mathcal{D} = \text{merge}(\mathcal{N}_i, \mathcal{N}_j)$ ;

- vi. Replaces the contents of the *global buffer* with  $\mathcal{D}$ ;
- vii. Increments the global model weight, i.e.,  $\phi_i += 1$ .

Given the contents of the static dataset  $\mathcal{D}$  may contain prototypes from local and global codebooks, each prototype  $\mathbf{w}_l \in \mathcal{D}$  is effectively a tuple  $\langle \mathbf{w}_l, \phi_l \rangle$ , where  $\phi_l$  is the weight from the prototype's originating model. During the merge procedure, when two overlapping prototypes are detected, the one with the highest weight is kept.

Consequently, and to account for different weights of prototypes, the global UbiSOM learning rule is modified to scale the prototype adjustments by the weighting factor of the current prototype — Eq. (6.5), where the function  $\phi(\mathbf{w}_l)$  returns the associated weight of the prototype from the previous tuple.

Let  $\mathbf{x}(t) \equiv \mathbf{w}_l \in \mathcal{D}$

$$\phi_{max} = \max\{\phi(\mathbf{x}(t)) : \forall \mathbf{x} \in \mathcal{D}\}$$

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \frac{\phi(\mathbf{x}(t))}{\phi_{max}} \eta(t) h'_{ck}(t) [\mathbf{x}(t) - \mathbf{w}_k(t)] \quad (6.5)$$

The magnitude of the updates is therefore additionally weighted (besides the weighting imposed by learning parameters) by the normalized weight of the prototype in the  $]0, 1]$  interval. Also, as in the distributed methodology, attending to the fact that the global model is learned from prototypes of data, the distribution is assumed stationary, i.e., prototypes are drawn randomly from the global buffer.

It should be noted that the local codebooks can be simultaneous and seamlessly used in the distributed collaborative learning setting.

## 6.4 Experimental Evaluation

In this section the above methodologies are applied to an illustrative problem using artificial data. Given the general methodology relies mainly on the UbiSOM algorithm already evaluated in the previous chapter, focus is given on the quality of global models obtained through either *distributed* or *collaborative* learning settings. Also, the differences between filtered and unfiltered codebooks is illustrated, as well as the effect of the  $r_w$  parameter.

### 6.4.1 Artificial Data

The *same* cluster structure described by the *Complex* artificial data stream was used to evaluate the methodologies. However, the two-dimensional input space was split into four quadrants from where separate data streams were generated, each containing 100 000

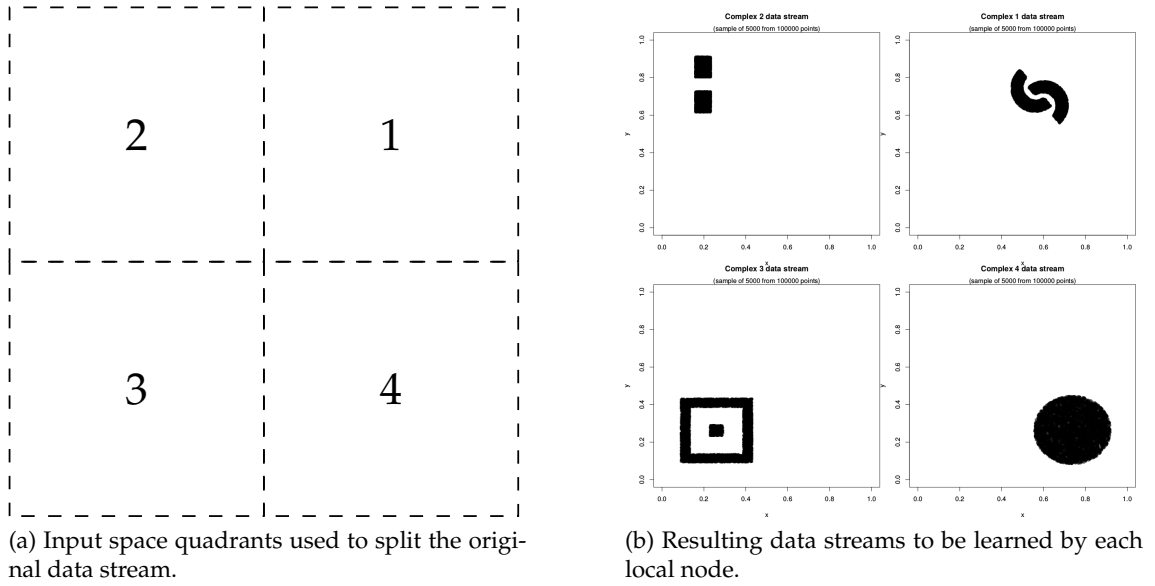


Figure 6.5: Generation of distributed data from the *Complex* artificial data stream.

observations. This generation process is depicted in Figure 6.5 and we effectively obtain four data streams pertaining the same shared problem. The goal is for each of these individual data streams to be learned by the local UbiSOM model of a processing node, while performing collaboration among nodes and/or centralizing the codebooks for a global model creation. In this particular scenario, each node is learning a stationary distribution.

#### 6.4.1.1 Simulated Environment and Parameterization

Four distinct and collaborating nodes are simulated in this environment, labeled  $Node_1$  to  $Node_4$ , each learning the respective data stream from its *Complex* quadrant. A *reference* node learns the original *Complex* data stream for comparison purposes. The same environment is utilized to evaluate both learning strategies.

The sequence of events in the simulation is schematized in Figure 6.6. Each depicted “time lapse” is a random value generated between  $[70, 90]$  seconds. These values do not have any specific meaning besides triggering occasional collaborations between nodes, simulating encounters in a physical space. After the environment is created, each node  $i = \{1, 2, 3, 4\}$  starts streaming observations from its data stream, learned by the respective local UbiSOM model. During the simulation, collaborations between nodes are triggered after random time lapses, simulating the establishment of opportunistic (peer-to-peer) connections between the nodes, e.g., proximity-based, where the codebooks  $\mathcal{N}_i$  are exchanged and merged to populate the individual *global buffers*, from where the global UbiSOM models draw the weighted learning observations. At the end of the simulation the local filtered codebooks  $\mathcal{L}'_i$  are centralized to produce a global Batch SOM model.

The sequence of collaborations between the nodes is the set  $\{\{1, 2\}, \{1, 4\}, \{1, 3\}\}$ .

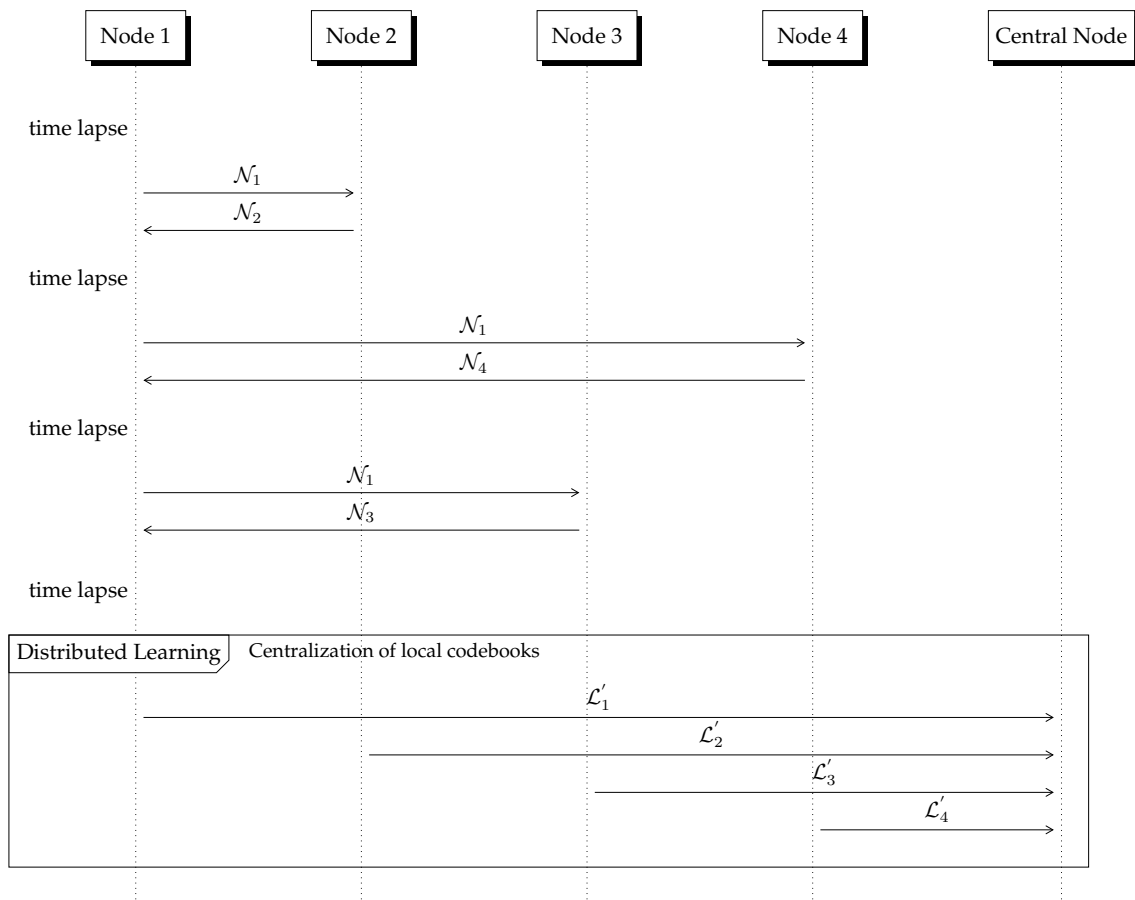


Figure 6.6: Sequence of events in the simulated ubiquitous environment.



		Local/Global UbiSOM		Centralized Batch SOM	
		Lattice size	$20 \times 40$	Lattice size	$20 \times 40$
Simulation Environment		$\eta_i$	0.1	$\eta_i$	0.1
Number of nodes	4	$\eta_f$	0.08	$\eta_f$	0.01
Duration	400 <i>sec</i>	$\sigma_i$	0.6	$\sigma_i$	$1/2\sqrt{20^2 + 40^2}$
Random time lapse	{70, 90} <i>sec</i>	$\sigma_f$	0.2	$\sigma_f$	1
Data stream speed	200 <i>obs/sec</i>	$\beta$	0.8	Order epochs	10
		$T$	1500	Tune epochs	40
		Other			
		$r_{\mathbf{w}}$	0.01		

Table 6.2: Parameters used in the simulated ubiquitous environment.

Note that only  $Node_1$  collaborates with the other 3 nodes in the attempt to learn the entire *Complex* distribution. After each collaboration, the codebook  $\mathcal{G}_1$  should cumulatively describe the learned distribution, while each  $\mathcal{G}_i$ ,  $i \neq 1$  should learn its portion of the distribution plus what  $\mathcal{G}_1$  can already describe at that time. Ultimately,  $\mathcal{G}_3$ , after the last collaboration between  $Node_3$  and  $Node_1$ , should approximately also describe the entire distribution, given  $Node_1$  has previously collaborated with  $Node_2$  and  $Node_4$ .

The parameterization of the experiment is summarized in Table 6.2. The environment simulates 4 collaborating nodes, each possessing a local and global UbiSOM model with the same parameters; a “reference” local model learns the whole *Complex* data stream, for comparison purposes. For this experimental evaluation the UbiSOM parameters obtained in the PSA for the *Complex* data stream (see Section 5.5) were used. Each node processes a learning example at a rate of 0.05 seconds per sample. This yields a presentation rate of the observations to the UbiSOM algorithms in the order of 500 observations per second, where the UbiSOM algorithm is able to converge to a stable state of the distribution in  $\approx 5$  seconds (this also derives from the UbiSOM PSA). The threshold value of  $r_w$  was set to 0.01. Given the normalization applied to distances in the *merge* procedure, this eliminates a duplicate prototype if another lies within 1% of the input space largest diagonal. The centralized Batch SOM parameters were set as in Chapter 4, namely for the offline SOM models of Section 4.5.5.

It should be noted that “time” is relative in this experiment. Simulation times and/or data stream rates can be increased/decreased, but it would not alter the objective of the experiment.

#### 6.4.1.2 Collaborative Learning

The simulated environment was run ten times, from which an illustrative run is presented, and the mean quantization error of the resulting models averaged for comparison purposes. Table 6.3 shows the simulated sequence of collaborations together with

the new contents of the *global buffer*, i.e., the resulting dataset  $\mathcal{D}$  from the collaboration procedure; these highlight the codebook *merge* procedure and prototype weighting. Figures 6.7 and 6.8 depict the obtained models and corresponding U-Matrices for all nodes, respectively.

The above results must be analyzed together, given their complementarity. Note that, following the simulated sequence of collaborations, more precisely  $\{\{1, 2\}, \{1, 4\}\}$ ,  $\mathcal{G}_2$  and  $\mathcal{G}_4$  also depict approximately  $\mathcal{G}_1$  short after each of those collaborations. Given each local UbiSOM is learning a stationary data stream, these models do not change during the simulation, after convergence. In Figure 6.7 we can observe that they are successful in modeling and exhibiting the underlying cluster structure. In particular, the U-Matrices in Figures 6.7a, 6.7b and 6.7c indicate 2 clusters in each model, while in Figure 6.7d no cluster structure is visible indicating a single cluster. On the other hand, the global UbiSOM models are obtained collaboratively and may change over time.

Pertaining the following analysis of the depicted simulation, recall that each UbiSOM model is  $20 \times 40$  in size, effectively containing 800 prototypes. In the first simulated collaboration between  $Node_1$  and  $Node_2$ , at time 95 *sec*, the respective global models do not contribute with any prototypes at this time because the *global buffers* are empty. Consequently,  $\mathcal{N}_1 = \mathcal{L}'_1$  and  $\mathcal{N}_2 = \mathcal{L}'_2$ , i.e., they only contain the filtered codebooks from the local models. No overlapping occurs between  $\mathcal{N}_1$  and  $\mathcal{N}_2$ , resulting in a dataset  $\mathcal{D}$  with 1194 (from possible maximum of  $2 \times 800 = 1600$ , consequence of the codebook filtering) weighted prototypes with  $\phi = 1$ ; this forms the content of the *global buffer* of these nodes — these prototypes then start being fed cyclically to the respective global UbiSOM models. After convergence, the obtained global model can be visualized in Figure 6.8b. The U-Matrix clearly indicates 4 clusters, i.e., the cumulative distribution of quadrants 1 and 2, also allowing to infer the relative distances between individual clusters. This simultaneously shows the global model  $\mathcal{G}_1$  before the next collaboration and  $\mathcal{G}_2$  until the end of the simulation.

After 177 *sec*, the second collaboration between  $Node_1$  and  $Node_4$  is triggered. This time, the resulting dataset  $\mathcal{D}$  now contains prototypes from  $\mathcal{L}'_1$ ,  $\mathcal{G}'_1$  and  $\mathcal{L}'_4$ . It should be pointed out that  $\mathcal{D}$  only contains 1220 prototypes (from the possible  $3 \times 800 = 2400$ ), because  $merge(\mathcal{L}'_1, \mathcal{G}'_1)$  removes most of the overlapping prototypes, considering the global model  $\mathcal{G}'_1$  already describes well the local distribution. However, some are not removed, allowing  $Node_1$  to “reinforce” its locally learned observations in the transferred set  $\mathcal{N}_1$  (this is also visible in the next collaboration). Prototypes originating from  $\mathcal{G}'_1$  are weighted with  $\phi = 1$ , while the ones originating from  $\mathcal{L}'_1$  and  $\mathcal{L}'_4$  are weighted with  $\phi = 0.5$  — due to the fact that  $\phi(\mathcal{G}_1) = 2$  at this point in the simulation, being incremented afterwards. Following the same reasoning as before, Figure 6.8d depicts the resulting final global model of  $Node_4$ , simultaneously with the one from  $Node_1$  after this collaboration, where effectively the expected 5 clusters are visible.

Finally, the last collaboration between  $Node_1$  and  $Node_3$  occurs at 262 *sec*, with the

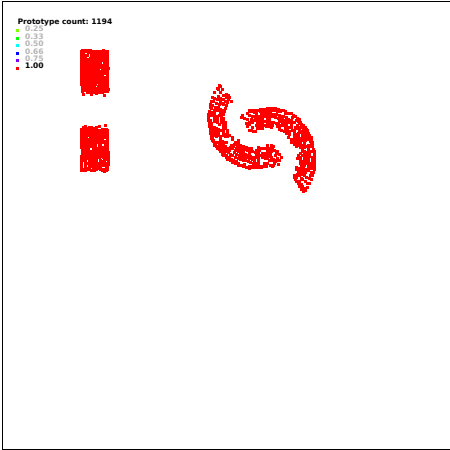
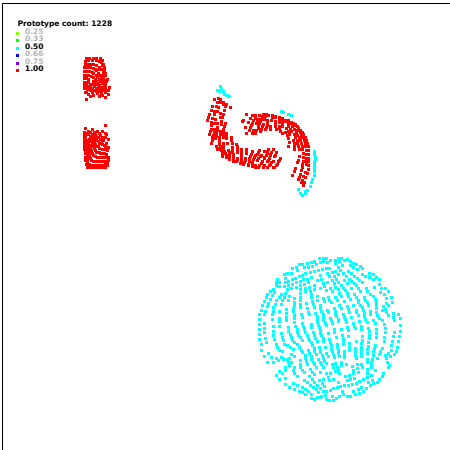
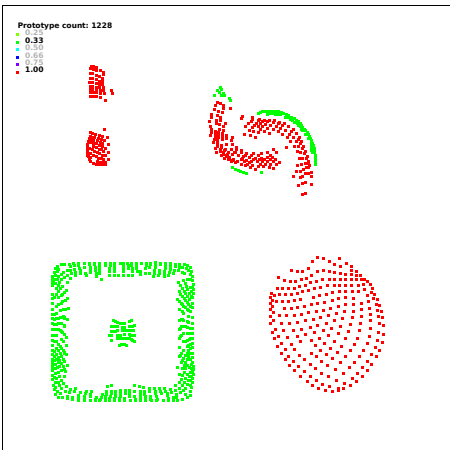
Time Lapse (sec)	Collaborating Nodes	Content of Global Buffer $\mathcal{D}$	$size(\mathcal{D})$
95	1 & 2		1194
177	1 & 4		1220
262	1 & 3		1220

Table 6.3: Sequence of collaborating events in the simulated ubiquitous environment, showing the contents of the obtained global buffer after each interaction.

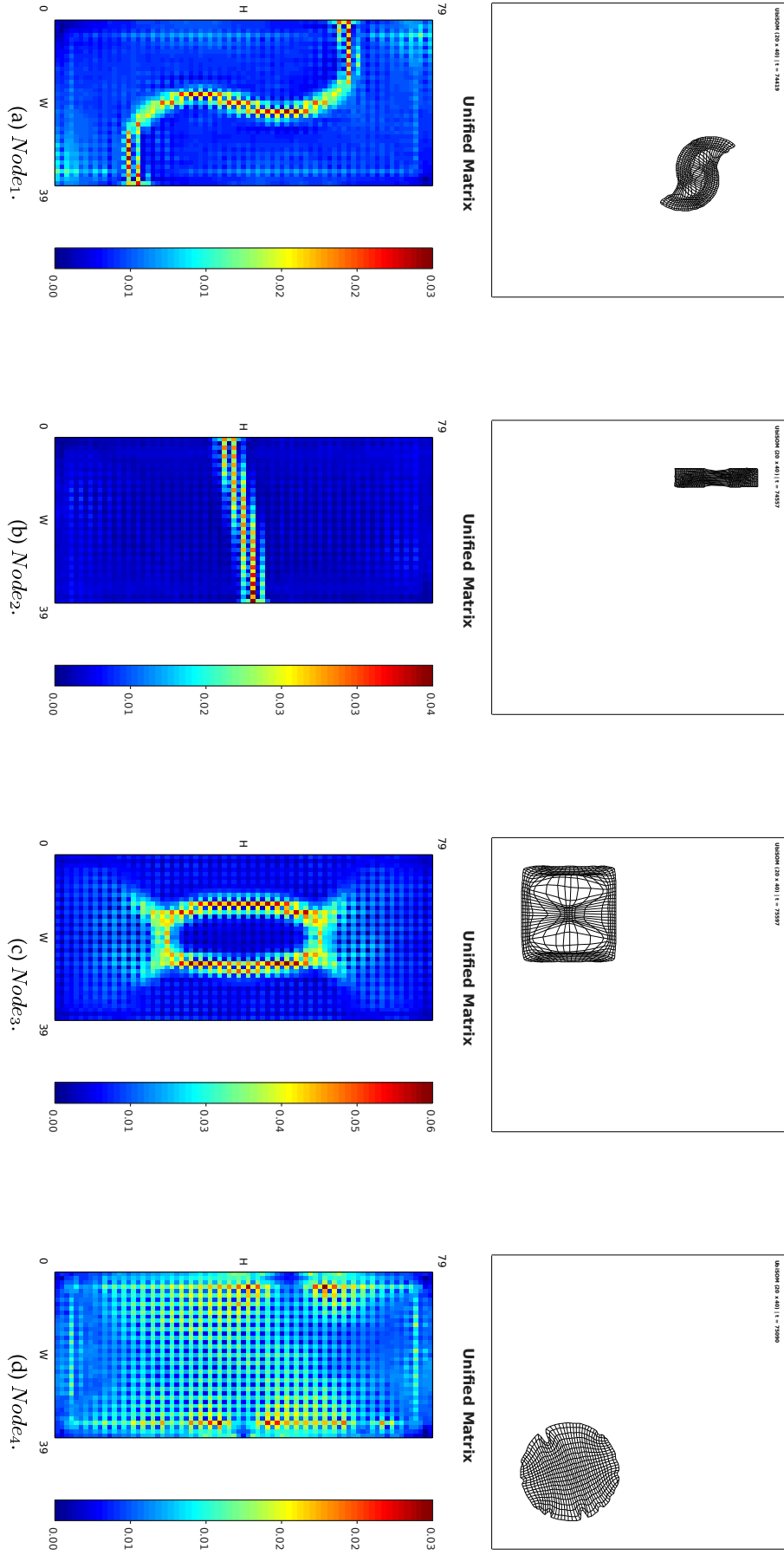


Figure 6.7: Final obtained local models in the collaborative learning methodology for the *Complex* distribution.

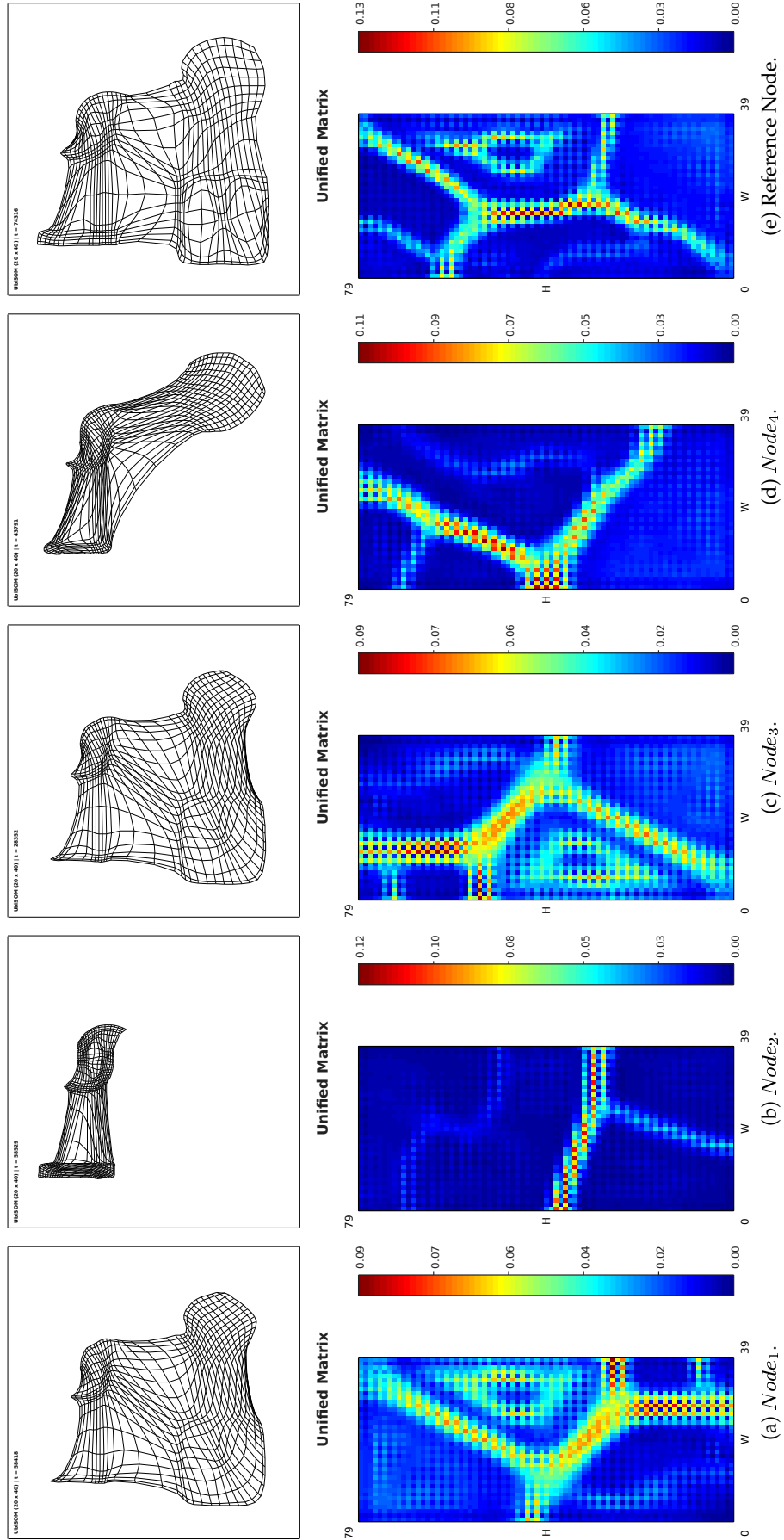


Figure 6.8: Final obtained global models in the collaborative learning methodology for the Complex distribution.

$i$	$\overline{QE}(\mathcal{G}_i)$	
1	1.5340e-2	$\pm 0.00124$
2	1.8478e-1	$\pm 0.012$
3	1.5159e-2	$\pm 0.00164$
4	7.9064e-2	$\pm 0.0096$
Reference	8.0608e-3	$\pm 0.00027$

Table 6.4: Normalized mean quantization error of each global codebook  $\mathcal{G}_i$  against the entire *Complex* data stream.

resulting dataset  $\mathcal{D}$  composed by  $\mathcal{L}'_1$ ,  $\mathcal{G}'_1$  and  $\mathcal{L}'_3$ . Again, from a maximum total of  $3 \times 800 = 2400$ , only 1220 weighted prototypes are obtained during the merge procedure, where prototypes originating from  $\mathcal{G}'_1$  are weighted with  $\phi = 1$ , while the ones originating from  $\mathcal{L}'_1$  and  $\mathcal{L}'_3$  are weighted with  $\phi = 0.33$  —  $\phi(\mathcal{G}_1) = 3$ , at this point. A “reset”, i.e., a transition back to the *ordering* state, occurred in the global UbiSOM of *Node*<sub>1</sub> after this collaboration (at  $\approx 274$  sec). This means that this global model, which had a lattice configuration similar to the one illustrated in Figure 6.8d, after the collaboration with *Node*<sub>4</sub>, was unable to adjust to the new distribution described by  $\mathcal{D}$  without a reorder or the prototypes. Figures 6.8a and 6.8c depict the final global models obtained for *Node*<sub>1</sub> and *Node*<sub>3</sub>, respectively. From both, the expected 7 clusters can be inferred in the U-Matrices. Note that the resulting lattices seem similar, but they differ in the rotation, which can be observed also in the U-Matrices. These models, compared to the *reference* model, are very similar, giving evidence of the feasibility of the proposed methodology.

After each of the ten runs of the simulated environment, the complete *Complex* data stream was projected against each global UbiSOM model of the collaborating and reference nodes; the mean normalized quantization errors were computed and averaged across runs. Results are presented in Table 6.4. The quantization capabilities of  $\mathcal{G}_1$  and  $\mathcal{G}_3$  are practically identical, indicating that the single collaboration between *Node*<sub>1</sub> and *Node*<sub>3</sub> allowed the later to learn the complete distribution with a single collaboration, while the first was able to learn it cumulatively. Compared with the value obtained for the *reference* node, they are very close. This is in line with the expectation that only an approximate model could be achieved through the collaborative methodology. However, from the previous illustrated results, the results from the visual exploratory analysis are identical, i.e., the detection of the 7 clusters.

#### 6.4.1.3 Distributed Learning

Using the same previous illustrative simulation, the complete centralized set composed by the individual filtered codebooks  $\mathcal{L}'_i$  in illustrated in Figure 6.9a. From the total  $4 \times 800 = 3200$  prototypes, only 2481 are centralized by the filtering mechanism performed at each node; the merge procedure did not find any duplicates. These prototypes

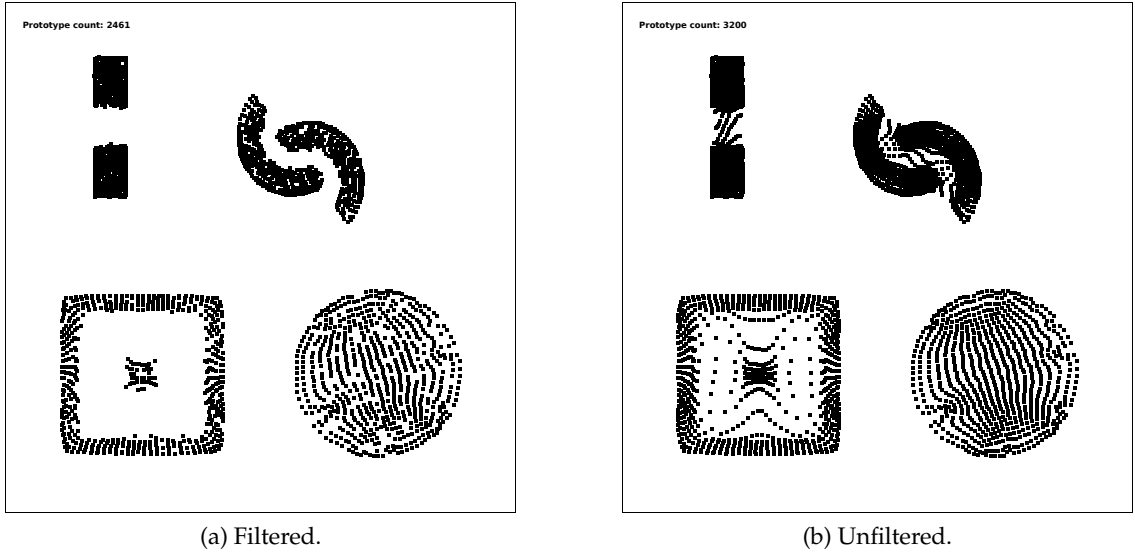


Figure 6.9: Filtered vs unfiltered centralized codebooks from individual nodes.

effectively describe the entire *Complex* distribution. By contrast, Figure 6.9b depicts the unfiltered version of this set, where several “unrepresentative” prototypes are observed. If these later prototypes were to be used as learning observations for the centralized model, this would degrade the obtained model; these prototypes would be given the same importance as the truly representative ones.

The resulting model and U-Matrix after running the Batch SOM algorithm is presented in Figure 6.10. The expected cluster structure of 7 clusters is visible.

## 6.5 Remarks and Future Work

In this chapter methodologies addressing the use of the UbiSOM algorithm in ubiquitous environments were presented. Proposed methods aimed at the current trade-off between local and global models. Local models can be seen as subjective views of the data, while global models more general and inclusive views of the problem.

Assuming each node on the network processes its own acquired data stream (pertaining the same problem/task), continuously maintaining a local UbiSOM model, two learning strategies were tackled — *distributed* and *collaborative*. The former consists in generating a one-shot global centralized SOM model from the current codebooks of a set of distributed nodes; in this setting the Batch SOM algorithm was used, for which, if scalability is a concern, parallel implementations are known, e.g., in (Silva and Marques 2007). The latter assumes nodes can move in the physical space and consists in collaboration between nodes through, e.g., proximity-based opportunistic networks. The proposed collaborative methodology allows each node to maintain a subjective view of the problem through its local model, while the global UbiSOM model evolves continuously with prototypes from different sources, namely the prototypes from its models and

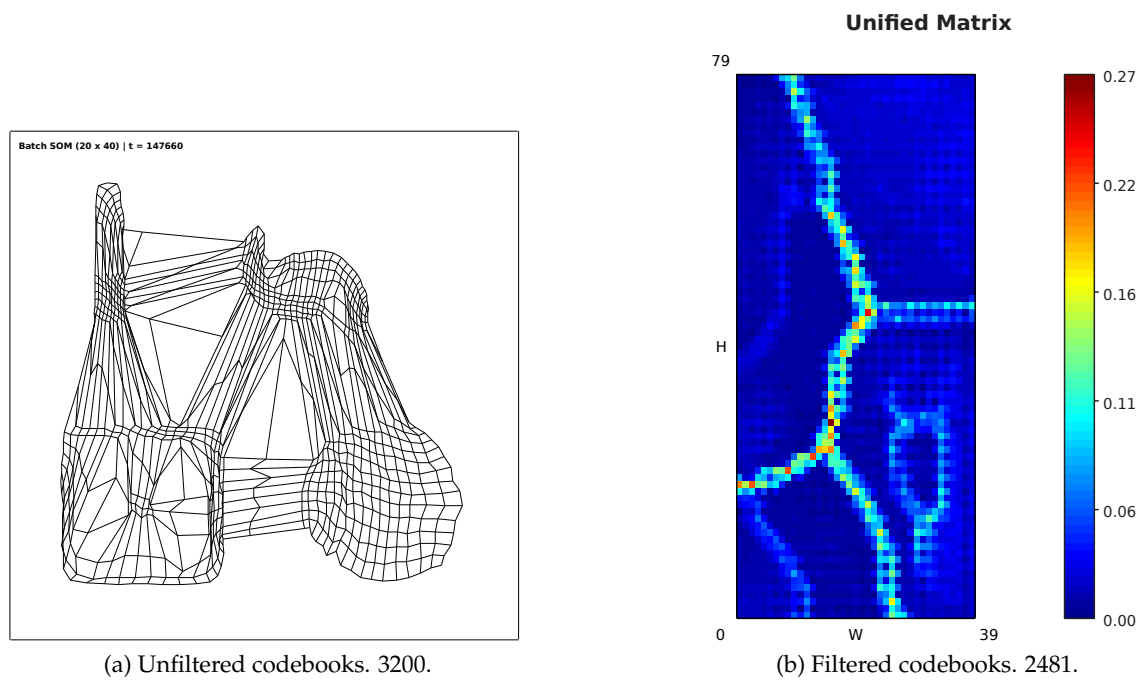


Figure 6.10: Centralized model and U-Mat.

from other nodes it encounters. This strategy resembles a cumulative learning mechanism, where “knowledge” from other sources is incorporated in the global model while retaining past learning. Both previous learning strategies rely on general techniques as codebook filtering of “unrepresentative” prototypes and codebook merging to eliminate “duplicate” prototypes.

The proposed methodologies were presented as initial attempts to the problematic of ubiquitous data streams. The *distributed learning* strategy is sound and simple in its conception — a real-world application can be found in Chapter 7. However, the *collaborative learning* strategy is more complex and less reliable. For example, it should be noted that with increasing collaborations performed by a node, its capability on modeling the cumulative distribution degrades slightly, consequence of the fixed amount of prototypes in the global model.

Regarding future work, some aspects are put forward in the following paragraphs.

**Change detection at local nodes.** An automatic change detection mechanism for the UbiSOM is still of interest in the proposed methodologies. For example, regarding the *distributed learning* strategy, if change is detected at a local node it may signal the central location that the current model has become obsolete and trigger the generation of a new global model. This is a typical strategy, e.g., in (Gama, Rodrigues, and Lopes 2011). However, this requires that prototypes from local nodes be kept separately, allowing them to



be replaced selectively.

In the *collaborative learning* strategy, at the moment, for a specific node, the local model is only incorporated in the global model when a collaboration procedure (with other node) is triggered. Automatic change detection over the local model would allow this incorporation to be made whenever change occurs, more precisely after the local UbiSOM has converged to the new distribution;

**Improvements for collaborative learning.** The presented methodology does not address any forgetting mechanism. Notwithstanding the usefulness of the methodology for collaborative learning in a cumulative sense, when taking into account the non-stationarity of local data streams things become more hazy, specially if one assumes nodes only communicate sporadically and when in proximity; then there is no way to know if prototypes obtained from a particular node have become obsolete, if it is never “seen” again. On the contrary, even if nodes are able to maintain continuous awareness of each other, then it requires that prototypes from other nodes be kept separately, allowing them to be replaced when change is signaled. However, this poses a problem in respect to the scalability of the approach, e.g., memory and communication overhead.

Additional future work considerations are made in the next chapter, dealing with real-world data, namely in Section 7.6.





# Real-World Applications

No amount of experimentation can ever prove me right; a single experiment can prove me wrong.

---

ALBERT EINSTEIN, GERMAN THEORETICAL PHYSICIST  
(1879-1955)

In the previous chapters, the StreamART2A/SOM methodology and the UbiSOM algorithm were presented and evaluated. Chapter 4 presented the StreamART2A algorithm as an *online* data stream abstraction procedure from where *offline* Batch SOM models are generated at user request. Chapter 5 presented the UbiSOM, capable of directly and continuously abstract non-stationary data streams. Chapter 6 transposed this capability to a possibly ubiquitous environment, where *distributed* and *collaborative* learning methodologies were proposed. Artificial data was used to perform the evaluations, given results then could be compared to expected outcomes. This chapter focuses on real-world application scenarios for the proposed algorithms and methodologies.

## 7.1 Chapter Overview

In this chapter, the proposed algorithms and methodologies are applied to three real-world problems — two involving sensor data (local and distributed) and one encompassing financial data. The versatility of SOM models obtained in a stream setting, regarding different data mining tasks, is also illustrated, namely, traditional clustering of observations, detecting correlated features and, by extension, the possibility to cluster the features themselves. The later, as discussed before, is intimately related to time series clustering, in a stream setting.

The chapter is organizing as follows. Section 7.2 provides motivation for the selected data, as well as the distinct aims that each application targets. In Section 7.3, an application over household electric consumption data stream is illustrated, while additionally presenting two new UbiSOM specific visualizations. In Section 7.4, an application using distributed air quality streams is presented, involving the generation of local and global models of pollutant patterns. In Section 7.5, the StreamART2A/SOM methodology is applied to the financial domain, with the overall purpose of grouping stocks that behave similarly along time. Finally, in Section 7.6, future work directions, specifically regarding real-world data, are put forward and discussed.

## 7.2 Motivation and Aim

Data streams are generated naturally within several applications as opposed to simple datasets. Such applications include network monitoring, web mining, sensor networks, telecommunications, and financial applications. All have vast amounts of data arriving continuously that can be mined for interesting and relevant information. Hence, being able to produce real-time models from where to perform data mining tasks assumes great importance within these applications. Another important characteristic of data streams is that they are often mined in a distributed fashion.

Publicly available real-world data streams may be difficult to obtain, depending on their nature, either because they may contain sensitive data that proprietors are not willing to and/or cannot share or because the data itself can be monetized. Limited by these constraints, of the three real-world scenarios addressed in the chapter, two involve publicly available data, while the third uses data provided by *GoBusiness Finance* (GoBusiness Finance 2016).

The aim of this chapter is the application of the proposed algorithms and methodologies utilizing real-world data, with the intent to demonstrate their relevance, usefulness and applicability. Sensor data and financial markets are both typical domains that generate data streams and illustrate possible applications of the proposed methods. A secondary aim is to validate the previously suggested UbiSOM parameter intervals in Section 5.5.3, with real-world data, i.e., to verify if the PSA results using real-world data coincide with the previous intervals obtained with artificial data.

## 7.3 Household Electric Power Consumption

In most cases, there is an inherent temporal component to the stream mining process. This is because data may evolve over time. This behavior of data streams is referred to as *temporal locality* (Aggarwal 2007), and we are interested in verifying the ability of the UbiSOM algorithm to model these continuous changes over time.

The first real-world application presented in this chapter involves a multidimensional

stream of sensor data related to household electric consumption. Due to the inherent nature of residential electricity consumption, this is an example of a challenging application where the UbiSOM may operate. Consequently, through such an application we are interested in extracting knowledge from the UbiSOM model that is maintained over time along the underlying data stream, e.g., clusters of patterns of usage, features values that contribute the most for the formation of those clusters (cluster descriptions) and correlated measurements. Additionally, two new UbiSOM specific visualizations are introduced, in order to provide the user with an additional visual confidence mechanism for the obtained UbiSOM models.

### 7.3.1 Data Description

Data is publicly available through the UCI repository (Lichman 2013), under the name “*Individual household electric power consumption Data Set*”, containing measurements of electric power consumption in one household with a one-minute sampling rate, over a period of almost 4 years, between 2007 and 2010. Although not clearly stated in the data source, it can be safe to assume that this house is located at Clamart (France), because this is the data’s author provided location. At Clamart, annual average temperatures ranged between minimums of  $2^{\circ}C$  and  $15^{\circ}C$  and maximums of  $8^{\circ}C$  and  $26^{\circ}C$  (between 2000 and 2012). This is important to provide a context to the electric consumption data. Disregarding date and time-related attributes, data is composed by the following attributes (features):

**Global active power:** household global minute-averaged active power (in *kilowatt*);

**Global reactive power:** household global minute-averaged reactive power (in *kilowatt*);

**Voltage:** minute-averaged voltage (in *volt*);

**Global intensity:** household global minute-averaged current intensity (in *ampere*);

**Sub-metering 1:** energy sub-metering No. 1 (in *watt-hour* of active energy); corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered);

**Sub-metering 2:** energy sub-metering No. 2 (in *watt-hour* of active energy); corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light;

**Sub-metering 3:** energy sub-metering No. 3 (in *watt-hour* of active energy); corresponds to an electric water-heater and an air-conditioner.

Note that  $(global\ active\ power * 1000 / 60 - sub\ metering\ 1 - sub\ metering\ 2 - sub\ metering\ 3)$  represents the active energy consumed every minute (in *watt-hour*) in the household by electrical equipment not measured in sub-meterings 1, 2 and 3.

Household Data Stream							
	Global.Active.Power (kW)	Global.Reactive.Power (kW)	Voltage (V)	Global.Intensity (A)	Sub.Metering.1 (W/h)	Sub.Metering.2 (W/h)	Sub.Metering.3 (W/h)
Min.:	0.076	0.000	223.2	0.200	0.000	0.000	0.000
1st Qu.:	0.308	0.048	239.0	1.400	0.000	0.000	0.000
Median:	0.602	0.100	241.0	2.600	0.000	0.000	1.000
Mean:	1.092	0.124	240.8	4.628	1.122	1.299	6.458
3rd Qu.:	1.528	0.194	242.9	6.400	0.000	1.000	17.000
Max. :	11.122	1.390	254.2	48.400	88.000	80.000	31.000
NAs:	25979	25979	25979	25979	25979	25979	25979
Total: 2 075 259							
Complete: 2 049 280							

Table 7.1: Data summary for UCI’s “Individual household electric power consumption Data Set”.

Table 7.1 provides a 5-number data summary that enables us to grasp the characteristics of the data. Here we can observe the individual ranges, means, 1<sup>st</sup> and 3<sup>rd</sup> quartiles and missing value count per feature. For example, if we look at the sub-metering values all have a 1<sup>st</sup> quartile of 0 and median values of 0 or 1, meaning that practically 50% of the time no consumption is measured there. This immediately shows that electricity is not consumed continuously, but sporadically.

Missing values occur simultaneously for all features, where some type of failure may have occurred, e.g., power outages or sensor maintenance. For example, over one-day spans of missing data occur at:

- 2007-04-28 00:21:00 — 2007-04-30 14:23:00
- 2009-06-13 00:30:00 — 2009-06-15 07:34:00
- 2010-01-12 14:53:00 — 2010-01-14 19:01:00
- 2010-03-20 03:52:00 — 2010-03-21 13:38:00
- 2010-08-17 21:02:00 — 2010-08-22 21:27:00
- 2010-09-25 03:56:00 — 2010-09-28 19:12:00

For the generation of the *Household data stream*, date and time related features were discarded and only complete observations were kept, resulting in a total of 2 049 000 observations, with  $d = 7$ . All observations were normalized in the unit hypercube  $[0, 1]^d$ .

### 7.3.2 Parameterization

The PSA was performed using only data from 2007 (Figure 7.1) and is summarized in Table 7.2; the plot allows us to confirm the various trends in the electric consumption. We can see that the best parameterization, i.e.,  $T = 2000$  and  $\beta = 0.9$  is within the intervals provided in Section 5.5.3 using artificial data, namely  $\beta \in [0.6, 0.9]$  and  $T \in [1000, 2500]$ . The value of  $T = 2000$  suggests that, given the numerous changes along consumption patterns, a relatively larger window provides better stability/convergence throughout the learning process. Also,  $\beta < 1$  is in line with previous findings that confirm the added value of this metric, as opposed to using only the *average quantization error*, i.e., with  $\beta = 1$ .

For this illustrative application, the general parameters of the UbiSOM remained the same as in the previous chapters, i.e., map size of  $20 \times 40$ ,  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$ ,  $\sigma_f = 0.2$  and  $(T = 2000, \beta = 0.9)$  from this PSA.

### 7.3.3 The BMU Map and Activity Map Visualizations

Before we proceed, it should be reinforced that an UbiSOM model does not reflect only a particular instant in time in the stream, but a summarization of recently learned data.

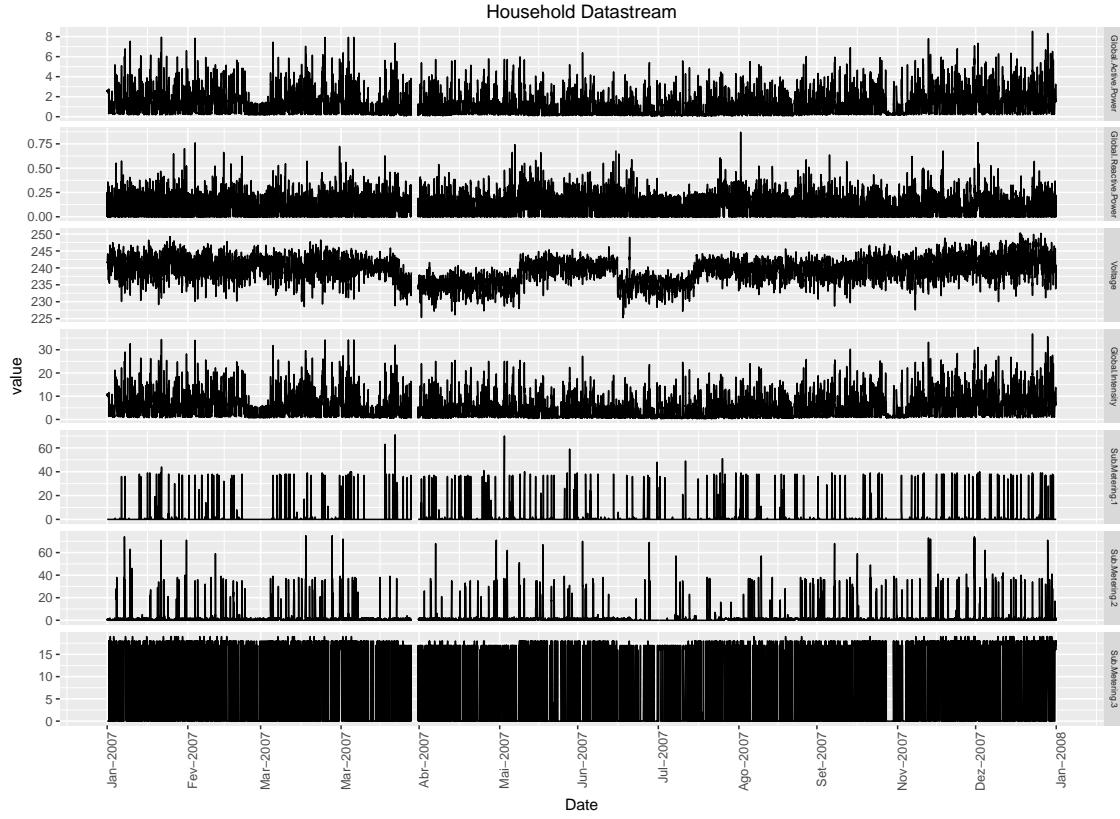


Figure 7.1: Household data stream in the year 2007.

Data stream	$T$	$\beta$	Mean $E_q^l(t)$	Mean $\lambda(t)$	Mean $TE(t)$	Resets
<i>Household</i>	2000	0.9	1.7875e-2	9.9619e-1	1.3155e-2	0
	2500	0.7	1.8308e-2	9.9868e-1	1.3782e-2	3
	2500	0.8	1.8190e-2	9.9841e-1	1.3749e-2	0

Table 7.2: Optimization results for a PSA over the *Household* data stream (only with data from 2007).



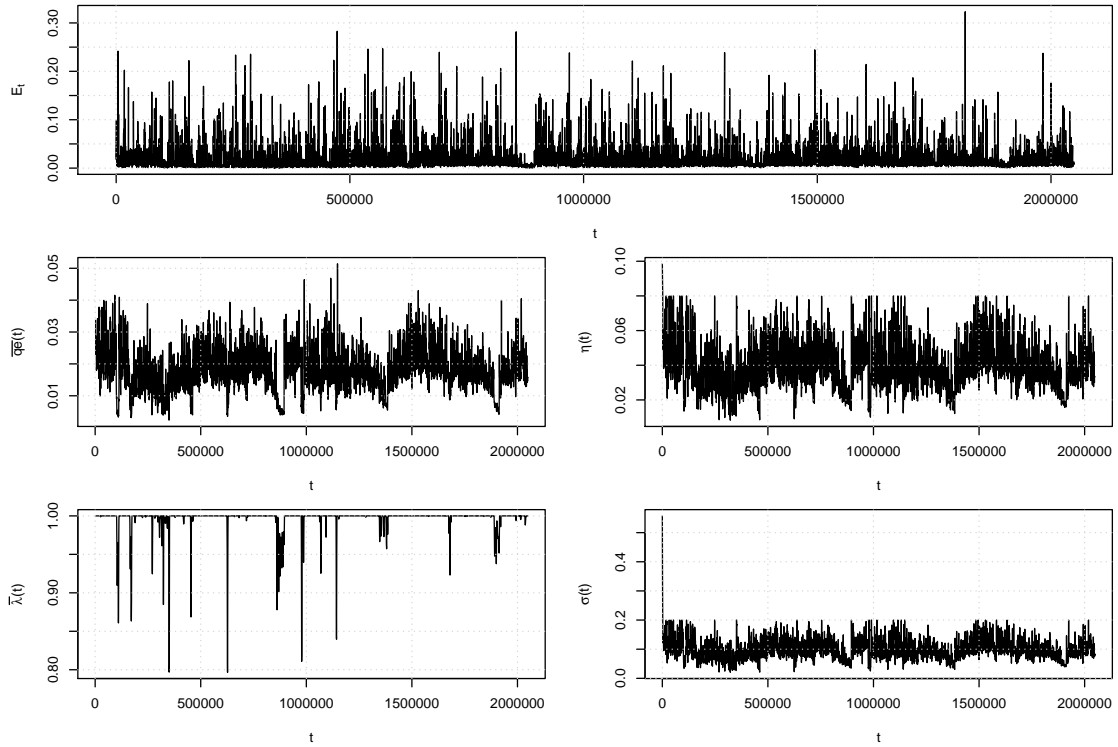


Figure 7.2: Assessment metrics and learning parameters evolution in the *Household* data stream.

Also, the term "recent" must be interpreted with care, taking into account the non-stationarity of the underlying distribution. If the distribution is in a stable state, then the model may represent a longer period of data. By contrast, with evolving data the period is shorter because the learning parameters will force the model to give more importance to recent observations. Even so, there are areas in the map that may still represent older information if newer data does not require a large adjustment of the lattice.

As opposed to the StreamART2A/SOM ensemble, where the SOM model is generated for a particular time horizon, the current UbiSOM abstraction is dependent on the order of the observations and underlying changes over time. Therefore, besides the trend of the drift function  $d(t)$  to suggest the degree of confidence in the current UbiSOM model, two new visualizations are introduced in this section as a means to provide a temporal visual reference of the current UbiSOM abstraction, namely using the extended information contained in each UbiSOM neuron pertaining the BMU and update *timestamps*, i.e.,  $t_k^{bmu}$  — *BMU Map*, and  $t_k^{update}$  — *Activity Map*. In these visualizations, the *timestamps* are color-coded, similarly to the component planes, i.e., "warmer" colors indicate more recent *timestamps*, while "cooler" colors indicate older ones.

### 7.3.4 Application Results

The entire learning process, i.e., the evolution of the averaged metrics and learning parameters, is depicted in Figure 7.2, from where we can see that indeed the underlying

distribution changes frequently along time. Please be aware that the plots depict more than two million iterations; at much smaller intervals, the behavior of the assessment metrics and learning parameter estimation is smoother.

During learning over the *Household* data stream two snapshots of the UbiSOM model were extracted at:

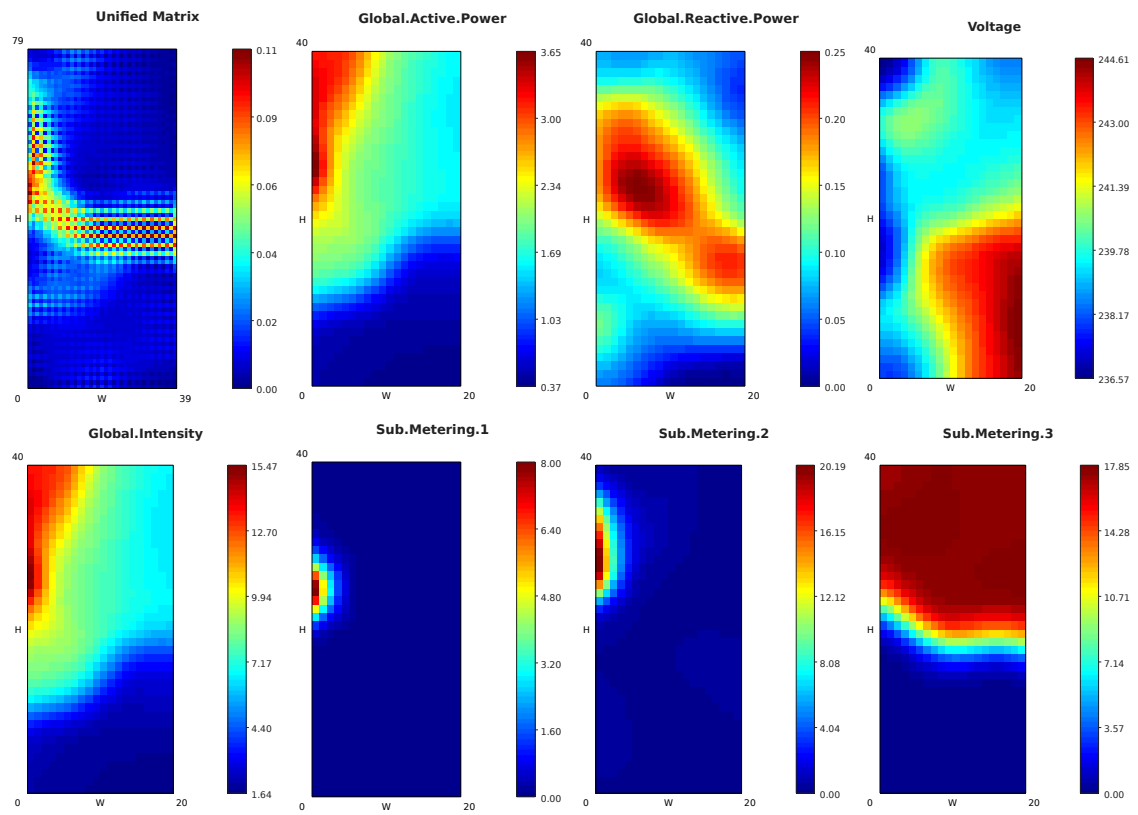
- i.  $t = 586\,861$ , analog to the date-time *2008-01-31 00:00*. In meteorological seasons, this date equates a mid-winter day and this model is referred hereafter as the *Household Winter model*;
- ii.  $t = 2\,049\,280$ , analog to the date-time *2010-11-26 21:02* (and to the last presented observation). This date corresponds to a mid-autumn day and the model would be referred to as the *Household Autumn model*.

The snapshots could be obtained at any time, taking care not to extract results while the algorithm is in the ordering state, but snapshots were taken shortly after the year used for the PSA (2007) and at the end of the data stream, coinciding with different meteorological seasons.

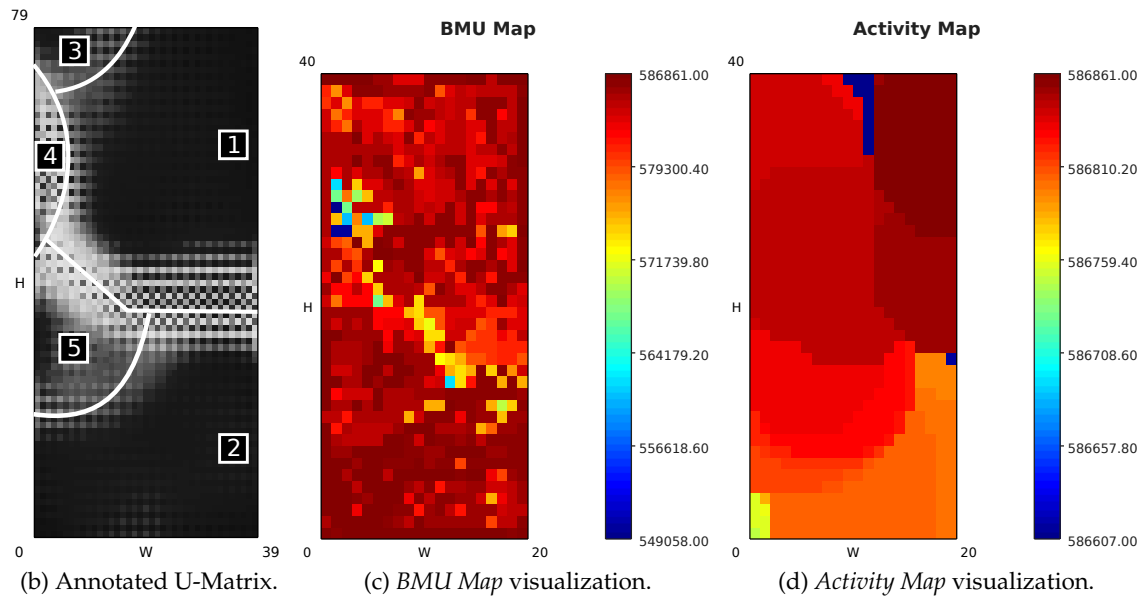
For the above two UbiSOM snapshots, the U-Matrix and component planes are presented; based on these two different visualizations, an annotated U-Matrix visualization is also presented. From these visualizations, an interpretation of typical knowledge that can be extracted from the obtained models is presented, concerning consumption patterns and correlations between measurements. Additionally, the *BMU* and *Activity Map* visualizations (see Section 7.3.3) are also presented and discussed. Finally, a depiction of the immediate observations preceding the snapshots is given, i.e., the prior 48h of data (2880 observations) regarding the selected dates, with the intent to help validate the inferred knowledge from the visualizations.

#### Household Winter model (at 2008-01-31 00:00) — Figure 7.3

- The standard visualizations, i.e., U-Matrix and denormalized component planes, for the snapshot are depicted in Figure 7.3a.
- Although, at least, two clusters are obvious in the U-Matrix, by analyzing it together with the component planes a more intricate cluster structure can be derived. Figure 7.3b depicts this U-Matrix with an annotated cluster structure. Together with the component planes they allow us to infer what conjunction of feature values contributes to the formation of those clusters, e.g.:
  - Clusters [1] and [2], the dominant ones, are composed by consumption patterns where sub-metering 3 attains high or low values, respectively;



(a) U-Matrix and component planes visualizations.



(b) Annotated U-Matrix.

(c) BMU Map visualization.

(d) Activity Map visualization.

Figure 7.3: Household Winter model and additional proposed visualizations.

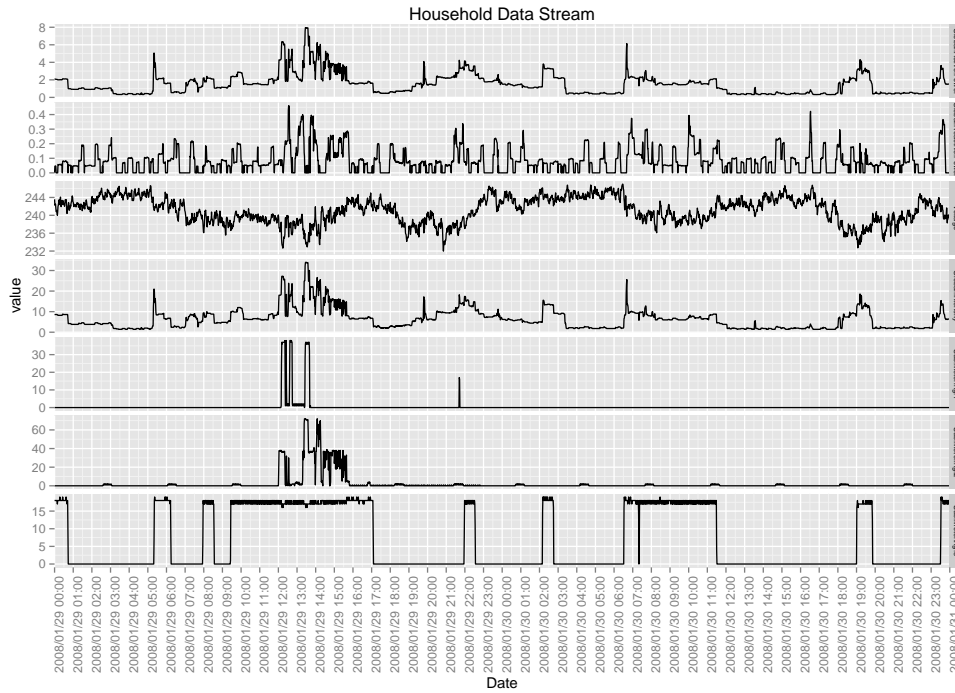
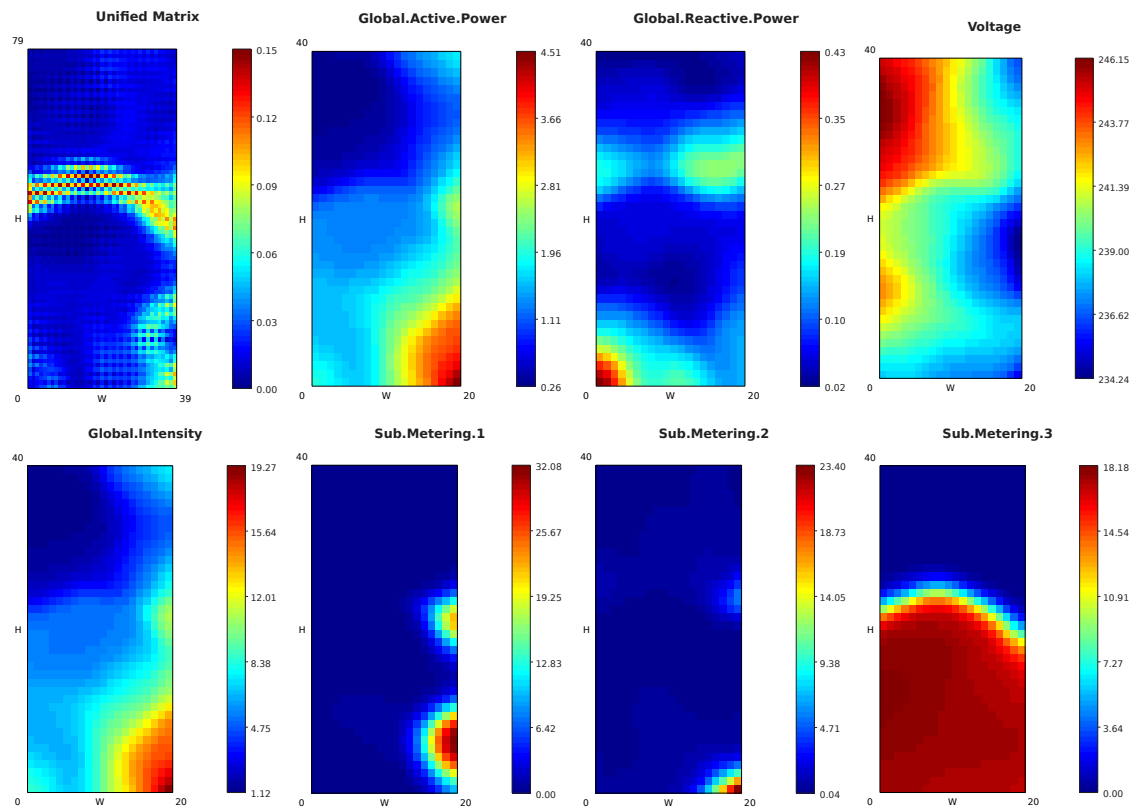
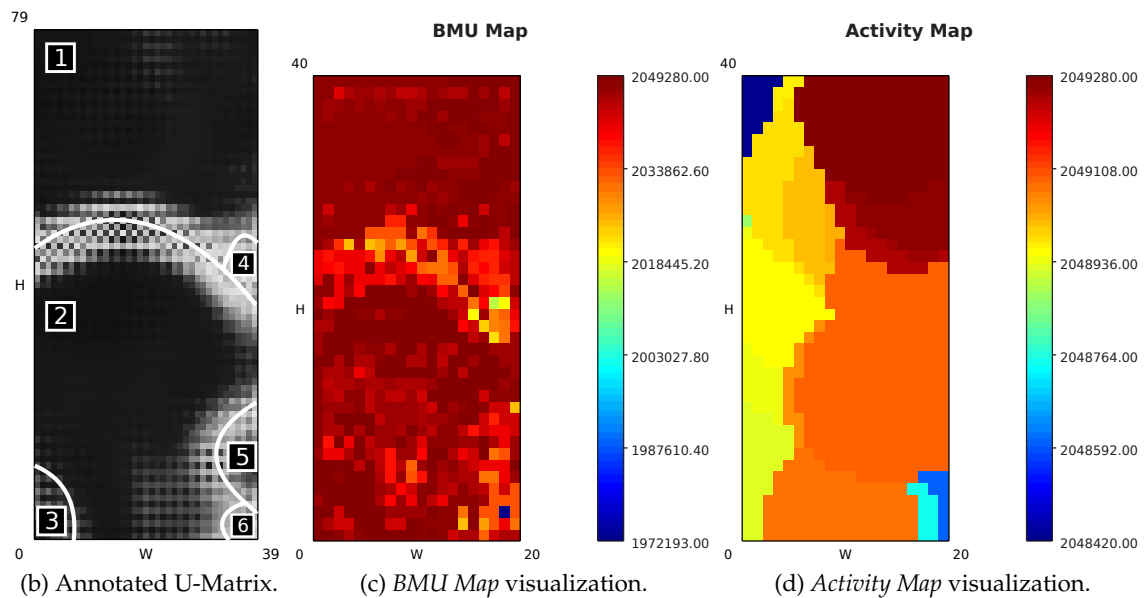


Figure 7.4: Preceding 48h of observations for the Household Winter model.

- By looking at the *sub-meterings* and *Global Active Power* component planes, we can infer that Cluster [3] contains consumption patterns where only other appliances (not measured by the sub-meterings) are in use;
  - Cluster [4] is characterized by consumption patterns with higher values in *sub-meterings* 1, 2 and 3. Given its relatively small size, this consumption profile is the least common in this model;
  - Finally, Cluster [5] identifies moderate consumption patterns for other appliances;
  - The above consumption clusters can be confirmed in the data stream plot of Figure 7.4.
- Regarding the BMU Map visualization (Figure 7.3c), it can be seen that the obtained model is expressing observations that span approximately 5 days of consumption patterns — the reasoning behind this statement comes from the fact that most depicted neurons have timestamps in the upper interval, i.e., between  $t \in [579\,300, 586\,861]$ , which divided by the daily observations (1440) results in  $\approx 5$  days. The rest of the prototypes that seem to be representing older information are actually “border” prototypes, i.e., they coincide with the cluster boundaries of the U-Matrix. In respect to the Activity Map (Figure 7.3d), we can see that the whole lattice has been recently updated; this visualization also allows us to observe the neighborhood kernel that is being used at this moment, although not the decreasing magnitude of the updates.



(a) U-Matrix and component planes visualizations.



(b) Annotated U-Matrix.

(c) BMU Map visualization.

(d) Activity Map visualization.

Figure 7.5: Household Autumn model and additional proposed visualizations.

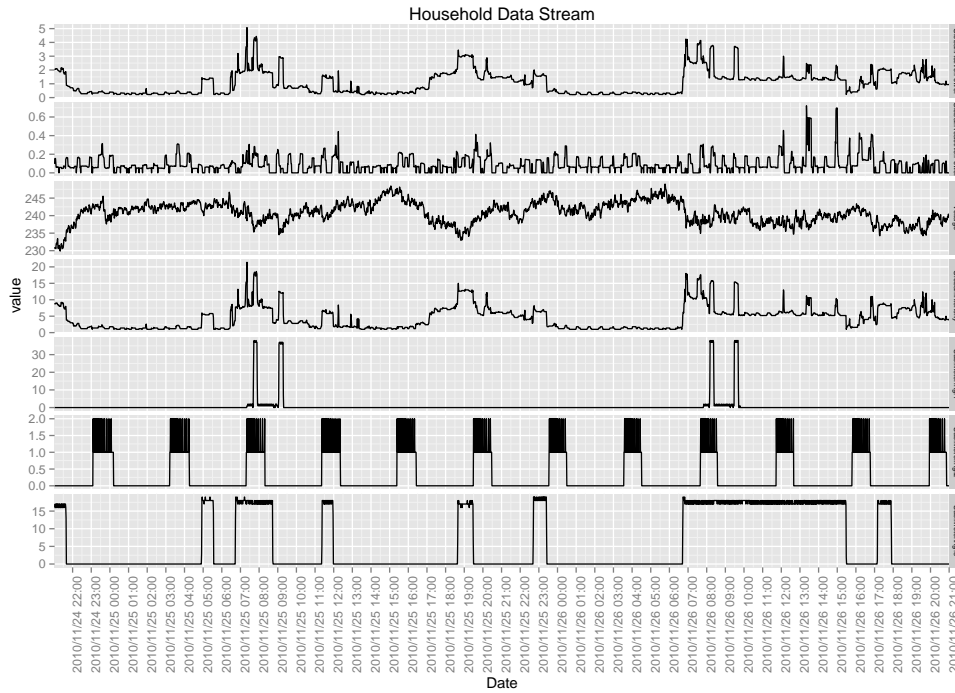


Figure 7.6: Preceding 48h of observations for the Household Autumn model.

#### Household Autumn model (at 2010-11-26 21:02) — Figure 7.5

- The standard visualizations, i.e., U-Matrix and denormalized component planes, for the snapshot are depicted in Figure 7.5a.
- Following a similar reasoning, Figure 7.5b depicts the U-Matrix with an annotated cluster structured, derived from the U-Matrix in conjunction with the component planes. From here, we can infer that, e.g.:
  - Again, Clusters [1] and [2] are formed by consumption patterns dominated by low or high *sub-metering 3* values, respectively. Note that between this snapshot and the previous the the UbiSOM lattice rotated in the hyperspace, while adapting to the underlying data stream;
  - Cluster [3] contains high consumption patterns in *sub-metering 3* when a higher amount of *global reactive power* is available;
  - Cluster [4] is characterized by consumption patterns with medium values where only *sub-metering 1* is active (the kitchen);
  - Cluster [5] identifies consumption patterns where *sub-meterings 1* and *3* are active;
  - Finally, Cluster [6] where *sub-meterings 2* and *3* are active;
  - The above consumption clusters can also be confirmed in the data stream plot of Figure 7.6.

- Similarly to the reasoning for the previous Winter model, the BMU Map visualization (Figure 7.5c) indicates the model is expressing approximately 7 days of consumption patterns, while older prototypes coincide again with cluster borders of the U-Matrix. The Activity Map visualization (Figure 7.5d) also indicates the whole lattice has been updated recently.

Comparing both models, the consumption profiles are not too different, and by inspecting the denormalized scale of the *Global Active Power* component plane, nor is the overall energy consumed. Other snapshots that were taken for summer days indicate different consumption patterns with substantially lower consumptions in *sub-metering* 3 (water-heater and air conditioning) and higher prevalence of active energy in *sub-metering* 2 (the laundry room). Active energy measured by *sub-metering* 1 (the kitchen) is always the least expressive for this household.

Regarding correlated measurements, the more obvious is the almost perfect positive correlation between the *Global Active Power* and *Global Intensity* features; this is expected and in agreement with the provided plots of the data streams (Figures 7.4 and 7.6). By comparison, voltage measurements seem to be negatively correlated to some degree. Reactive power can be seen as “excess power” and represents energy alternately stored and released by inductors and/or capacitors. It is not clear the relationship between this measurement and the others by looking at the models, nor in the data plots, for that matter.

This application illustrates the type of knowledge inference that the UbiSOM allows over a non-stationary multidimensional data stream, e.g., in monitoring applications. Similar applications can include multi-dimensional sensor data from any domain; in the next section, data streams of pollutant concentrations are addressed.

## 7.4 Distributed Air Quality Monitoring

This section illustrates another application of the UbiSOM and distributed learning methodologies to distributed air quality monitoring data in Portugal. The interest of this application is the generation of local models at single locations, from where knowledge about the prevalence of air pollutants can be obtained, which can then be centralized to produce a global model characterizing the area containing those locations.

*"Air quality monitoring is important for assessing the nature of the population exposure to air pollution. Assessment of population exposure is necessary for health impact assessment, which in turn is crucial for developing plans for air quality management and protecting the public health. (...) In general, exposure assessment requires both monitoring and modeling to identify the more problematic pollutants and identify target sources for reducing emissions and to implement an effective programme of air quality management for protecting human health."* — (Organization 1999).

Air quality monitoring is often used to determine the air pollution levels in urban or rural environments. A monitoring network produces concentration measurements that can then be compared with national and international guideline values. Portugal has a network of air quality monitoring stations deployed over the mainland and archipelagos (Azores and Madeira). This network is divided by established territorial zones and individual stations can be classified by a combination of their type of *environment* and type of *influence*:

**Type of environment:** **Urban** located in urban environments (cities);

**Suburban** located at the periphery of cities;

**Rural** located in rural environments.

**Type of influence:** **Traffic** monitors direct emissions from car traffic;

**Industrial** monitors direct emissions from industrial sources;

**Background** does not monitor direct emissions from any source; acts as a baseline, representing the pollution that any citizen, even living away from emission sources, is exposed to.

Portugal's mainland air quality monitoring network is illustrated in Figure 7.7, depicting the distribution of the stations, more concentrated in the coastal area, and their type based on the previous classification. In March of 2016 the network consisted of a total of 65 active monitoring stations, where 3 were located in the archipelagos. We should be aware that the monitoring stations are located at ground-level (the majority in altitudes lower than 300 m), consequently measuring ground-level pollution, with the intent of measuring pollution levels that directly affect the population.

Data from monitored pollutants is gathered at a hourly rate at each station, validated and sent to *QualAr*, the national air quality database of the *Agência Portuguesa do Ambiente* (APA). This agency then publicly discloses a daily Air Quality Index (AQI) for each region, forecasts and other statistical information. Reports on air quality are released annually.

The AQIs are of interest to us in this section, given we will reference them later when interpreting the obtained local and global models. Each station monitors a set of anthropogenic pollutants, generally related to their type of influence. The total number of pollutants measured along time, between all stations, since the network's inception, ascends to the dozens. Stations are added, upgraded or downgraded in respect to sensors, or deactivated throughout time. As a side note, removing sensors from a station may seem strange, but if over a relatively long period of time a specific pollutant never reaches significant concentrations, then no more efforts are employed at measuring that specific pollutant in that site. Notwithstanding a possible larger amount of monitored pollutants, only five are used to calculate the AQI in Portugal:

**Carbon monoxide** (*CO*) It is a product by incomplete combustion of fuel, natural gas, coal or wood. Vehicular exhaust is a major source of carbon monoxide;



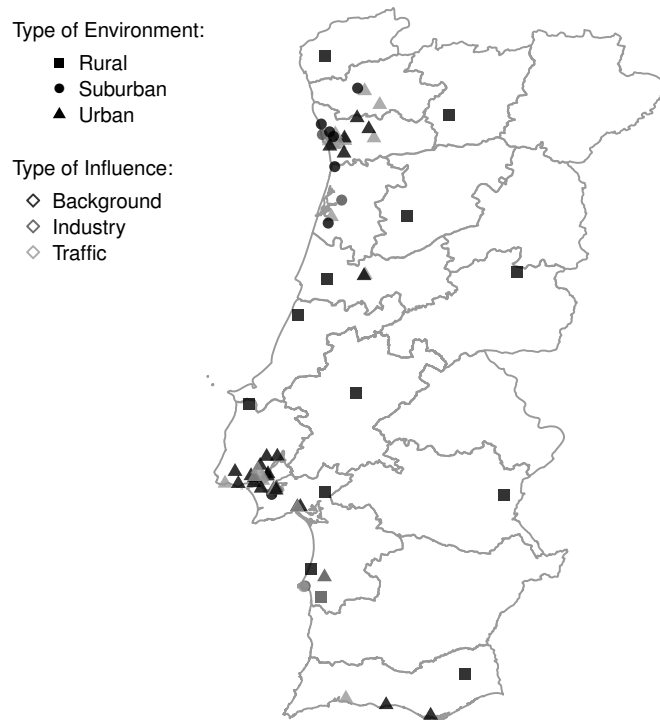


Figure 7.7: Location and characterization of air quality monitoring stations, regarding type of environment and influence, in Portugal.

**Nitrogen oxides** ( $NO_x$ ) Nitrogen oxides, particularly nitrogen dioxide ( $NO_2$ ), are expelled from high temperature combustion and one of the most prominent air pollutants;

**Ozone** ( $O_3$ ) Although essential in protecting us from UV radiation, ground level ozone is a pollutant and a constituent of smog. It can be formed from photochemical reactions with  $NO_x$  and  $CO$ , called ozone precursors;

**Particulate matter** ( $PM$ ) Regard fine particles suspended in the air. Human activities, such as the burning of fossil fuels in vehicles, power plants and various industrial processes generate significant amounts of particulate matter. Of interest are particles less than 10 microns in diameter — about  $1/7^{th}$  the thickness of the a human hair, and are known as  $PM_{10}$ .  $PM_{10}$  is among the most harmful of all air pollutants. When inhaled these particles evade the respiratory system's natural defenses and lodge deep in the lungs;

**Sulfur oxides** ( $SO_x$ ) Particularly sulfur dioxide ( $SO_2$ ), is produced in various industrial processes. Coal and petroleum often contain sulfur compounds, and their combustion generates  $SO_2$ . Derived compounds from reactions between  $SO_2$  and  $NO_2$  can lead to acid rain. This is one of the causes for concern over the environmental impact of fossil fuels.

Therefore, the AQI can be calculated for a particular area (agglomeration, zone or city)

	CO		NO2		O3		PM10		SO2	
AQI	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
Bad	10000	—	400	—	240	—	120	—	500	—
Weak	8500	9999	200	399	180	239	50	119	350	499
Medium	7000	8499	140	199	120	179	35	49	210	349
Good	5000	6999	100	139	60	119	20	34	140	209
Very good	0	4999	0	99	0	59	0	19	0	139

Table 7.3: Guideline values for the Air Quality Index (AQI) in Portugal. All values are in  $\mu\text{g}/\text{m}^3$ .

and uses averages between all measured values for the previous given pollutants in that area, i.e., for a single pollutant the values measured at all stations in the area of interest are obtained and averaged. The AQI always uses a minimum of 11 or 18 hourly averages to produce provisory (0h00 ~ 14h59) or definite (0h00 ~ 23h59) indices, respectively.

A qualitative AQI is obtained for each pollutant, according to Table 7.3. The AQI for a particular agglomeration, zone or city is determined by the worst index obtained among all pollutants. E.g., consider the following average values measured in an area:

- $SO_2$  —  $35 \mu\text{g}/\text{m}^3$  (Very good);
- $NO_2$  —  $180 \mu\text{g}/\text{m}^3$  (Medium);
- $CO$  —  $6000 \mu\text{g}/\text{m}^3$  (Good);
- $PM_{10}$  —  $15 \mu\text{g}/\text{m}^3$  (Very Good);
- $O_3$  —  $365 \mu\text{g}/\text{m}^3$  (Bad)

For this hypothetical area the AQI result is *Bad*, due to the observed concentrations for ozone ( $O_3$ ).

The dynamics between air pollution and atmospheric conditions are quite complex. Without delving into this subject, there are certain air pollutants levels that are worse in the summer and others that are worse in the winter. In the summer, ground level  $O_3$  levels are higher because more is produced due to photochemical reactions by UV radiation with  $NO_x$ . On the other hand, temperature inversions can transport the pollutants higher in the atmosphere. Also, generally, wind favors the dissipation of pollutants and rain can also help wash some particulates out of the air. Also, a rural area can experience higher levels of pollution than expected, depending of wind currents originating from more polluted areas.



Figure 7.8: Air quality monitoring stations in the district of Aveiro.

#### 7.4.1 Data Description

Data from Portugal's air quality monitor network is publicly available through *QualAr's* website ([QualAr website](#)). Data can be obtained by monitoring station, year and/or pollutant. Data is available from 1992 until the present day, but data for a particular year undergoes a validation procedure until October of the following year.

Some literature exists based on research using this monitoring data in the environmental domain. For example, in (Carvalho et al. 2010) the authors study a high ozone level anomaly at a location in northeast Portugal, by performing cluster analysis of patterns from the episodic days, which somewhat relates to our goals, but in a static data setting

To avoid extending the application into a larger number of stations, that would not contribute further regarding the intended demonstration, only data from the three stations in the district of Aveiro were used (Figure 7.8). The three stations, characterized by their types of environment and influence, are: *Aveiro* (urban/traffic), *Estarreja/Teixugueira* (suburban/industrial) and *Ílhavo* (suburban/background).

Given the need for homogeneous data streams to generate global models, unfortunately only the measurements for  $PM_{10}$ ,  $NO$ ,  $NO_2$  and  $NO_x$  were found in common at these stations. Hence, hourly data for these pollutants (the features) were gathered from the beginning of 2011 until the end of 2014. Table 7.4 provides a 5-number summary for each separate station's data. By this initial analysis we can initially observe that the station of *Aveiro* measures the higher levels of pollution, followed by *Estarreja/Teixugueira* and *Ílhavo*. We can also observe that missing values occur separately for individual pollutants.

A pairwise deletion of observations with missing values was performed to obtain complete and "synchronized" data streams presented to each local UbiSOM model. Figure 7.9 illustrates this distribution of missing values and the final proportion of obtained observations across all data streams. As a result, all observations in the final individual data streams correspond to the same sequence of dates, which are depicted in Figure 7.10

Aveiro Data Stream					Estarreja/Teixugueira Data Stream					Ilhavo Data Stream				
$PM_{10}(\mu g/m^3)$	$NO_2(\mu g/m^3)$	$NO(\mu g/m^3)$	$NO_x(\mu g/m^3)$		$PM_{10}(\mu g/m^3)$	$NO_2(\mu g/m^3)$	$NO(\mu g/m^3)$	$NO_x(\mu g/m^3)$		$PM_{10}(\mu g/m^3)$	$NO_2(\mu g/m^3)$	$NO(\mu g/m^3)$	$NO_x(\mu g/m^3)$	
Min.: 0.00	4.00	0.00	0.00		Min.: 0.00	0.00	0.00	0.00		Min.: 0.00	0.00	0.00	0.00	
1st Qu.: 20.00	14.00	0.00	11.00		1st Qu.: 13.00	6.00	2.00	10.00		1st Qu.: 14.00	2.00	0.00	1.00	
Median: 29.00	21.00	1.00	21.00		Median: 22.00	12.00	3.00	17.00		Median: 21.00	6.00	0.00	4.00	
Mean: 34.76	24.56	5.918	31.14		Mean: 28.46	16.06	8.05	28.43		Mean: 26.11	7.707	1.263	8.665	
3rd Qu.: 42.00	30.00	5.00	37.00		3rd Qu.: 36.00	22.00	7.00	34.00		3rd Qu.: 31.00	11.00	0.00	11.00	
Max.: 229.00	186.00	371.00	744.00		Max.: 200.00	110.00	214.00	381.00		Max.: 214.00	77.00	74.00	159.00	
NAs: 937	1182	1182	1182		NAs: 913	581	581	581		NAs: 1795	464	464	464	
Total: 35 064					Total: 35 064					Total: 35 064				
Complete: 33 112					Complete: 33 956					Complete: 32 958				

Table 7.4: Data summary of *QualAr* data obtained for Aveiro district air quality monitoring stations.

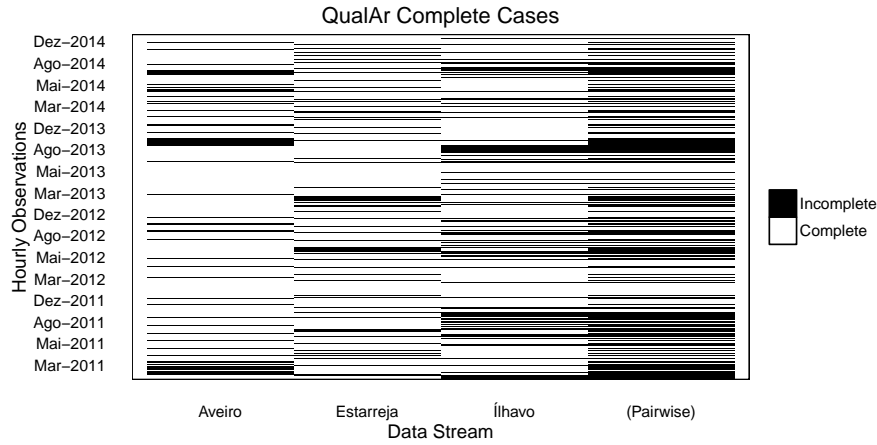


Figure 7.9: Obtained complete cases for the QualAr data streams.

— the *QualAr data streams* are named based on the originating station.

In order to retain the same feature ranges, data was normalized in the unit hypercube  $[0, 1]^d$ , but using the minimum and maximum values across all three data streams.

#### 7.4.2 Parameterization

Following the same aim relatively to the PSA with real-world data, Table 7.5 contains the summary of results obtained from the observations from the year 2014. Compared to the Household application, we observe that the best  $T$  values found are lower, e.g.,  $T = [500, 1000]$ . This may be explained by the fact that the variability of the data is lower than in the Household data stream, hence the balance between stability and convergence can be obtained with a lower  $T$  value. The common best parameterization found was used for all local UbiSOM models, i.e.,  $T = 1000$ ,  $\beta = 0.9$  and the general parameters kept, i.e., lattice sizes of  $20 \times 40$ ,  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$  and  $\sigma_f = 0.2$ . The Batch algorithm applied to the centralized codebooks had the same parameterization that in Section 6.4, namely map size of  $20 \times 40$ ,  $\eta_i = 0.1$ ,  $\eta_f = 0.01$ ,  $\sigma_i = 1/2\sqrt{20^2 + 40^2}$  and  $\sigma_f = 1$  with training enduring 10 and 40 ordering and tune epochs, respectively.

#### 7.4.3 Application Results

Three UbiSOM instances learned the separate data streams, i.e., *Aveiro*, *Estarreja/Teixugueira* and *Ílhavo*. Individual snapshots of the local models were taken at specific points in time and centralized to produce a global model; these instants are identified in the previous Figure 7.10:

- i.  $t = 4611$ , analog to the date-time 2011-08-31 23:00. In meteorological seasons, the date corresponds to a day at the end of summer. The resulting models taken at this time are referred to as the *QualAr Summer* models;
- ii.  $t = 7948$ , analog to the date-time 2012-01-31 23:00. This corresponds to a day in

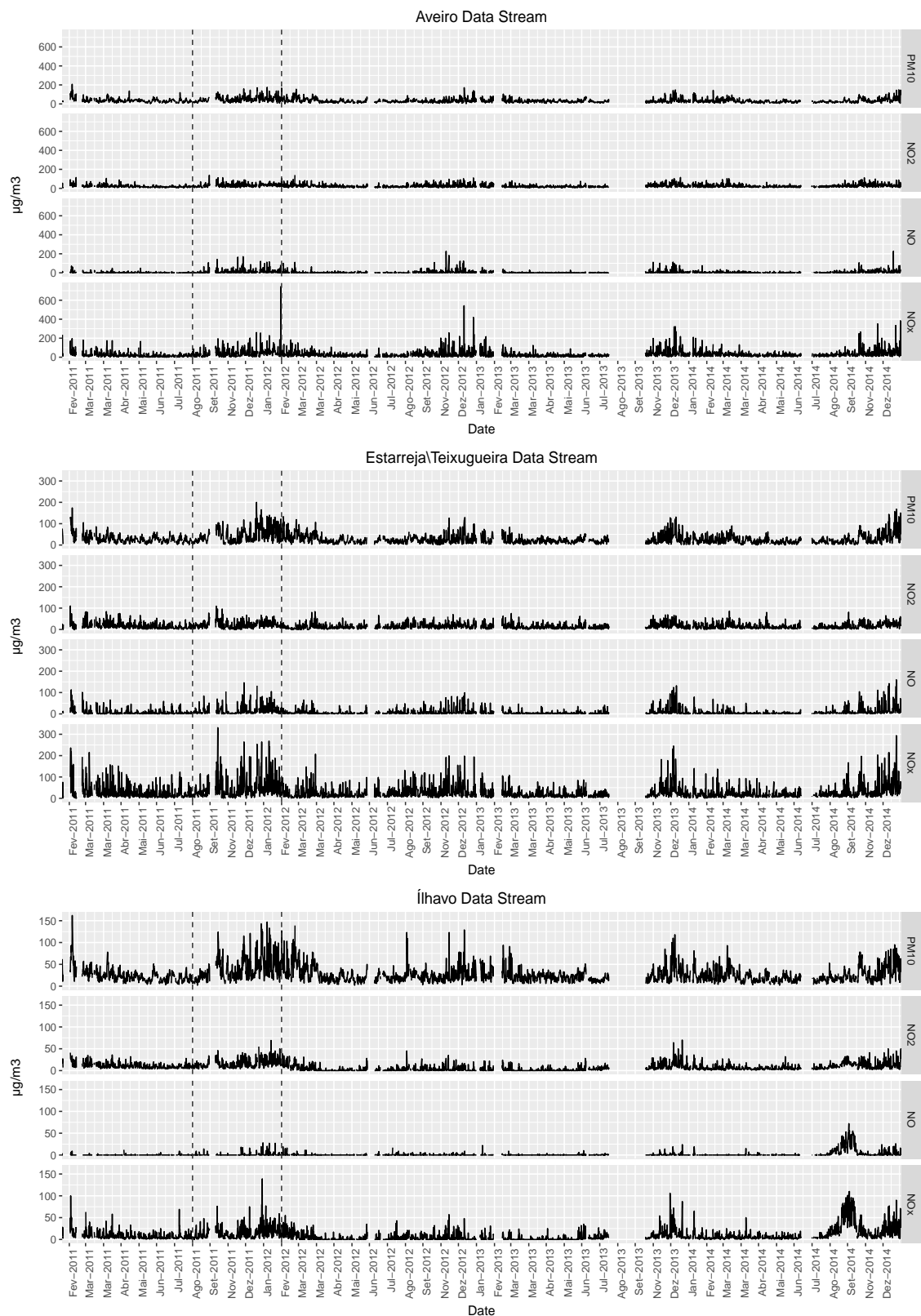


Figure 7.10: QualAr data streams after pairwise deletion of missing values.

Data stream	$T$	$\beta$	Mean $E'(t)$	Mean $\lambda(t)$	Mean $TE(t)$	Resets
Aveiro	500	0.9	1.0377e−2	9.6393e−1	5.1022e−3	0
	1000	0.9	1.0991e−2	9.8363e−1	5.1689e−3	0
	1000	1.0	1.0177e−2	9.4539e−1	4.6020e−3	0
Estarreja	500	0.9	1.5410e−2	9.5849e−1	5.2023e−3	0
	1000	0.9	1.5646e−2	9.7632e−1	5.5357e−3	0
	2000	0.8	1.6736e−2	9.9519e−1	5.4357e−3	0
Ílhavo	500	0.6	1.2992e−2	9.8436e−1	2.5978e−2	9
	500	1.0	1.0889e−2	7.2351e−1	1.8008e−2	1
	1000	0.9	1.2091e−2	9.7573e−1	2.9680e−2	2

Table 7.5: Optimization results for PSA analysis over QualAr data streams (only observations from 2014).

mid-winter and the models obtained are referred to as the *QualAr Winter* models.

These instants were chosen due to the larger continuous availability of prior data and with the intent to compare models from summer and winter seasons.

Figures 7.11 and 7.12 depict the visualizations, i.e., U-Matrix and denormalized component planes, derived from the local UbiSOM models and centralized Batch model obtained at the previous dates, respectively. The centralized *QualAr Summer model* was generated from 1190 filtered prototypes; the centralized *QualAr Winter model* from 1224 filtered prototypes (see Section ??).

In all local models, for example, it is obvious that there is a clear and expected positive correlation between the concentration patterns of the nitrogen oxides  $\{NO, NO_2, NO_x\}$  — with the component plane scales this may not be evident to the eye, but a closer inspection of the values show that  $NO_x$  concentration patterns vary directly with the combined concentrations of  $NO$  and  $NO_2$ . Given the limited number of pollutants modeled in this application, we cannot determine extensive and more interesting relationships between concentration patterns, e.g., no clear relationships can be inferred between  $PM_{10}$  and  $NO_x$  levels, using only these pollutants; their concentrations do not correlate positive or negatively, but exhibit some overlap intervals of simultaneous higher concentrations.

Nonetheless, and in comparison, pollutant concentrations are consistently higher in the *QualAr Winter* local models, which in turn is also translated to the respective centralized global model. Also, the local models indicate that the location that exhibits a higher average of pollution is *Aveiro*, followed by *Estarreja* and *Ílhavo* — this is consistent with the initial data summaries of Table 7.4. For the monitored pollutants, this is not surprising, consequence of vehicular exhaust. We should stress “for these pollutants” because

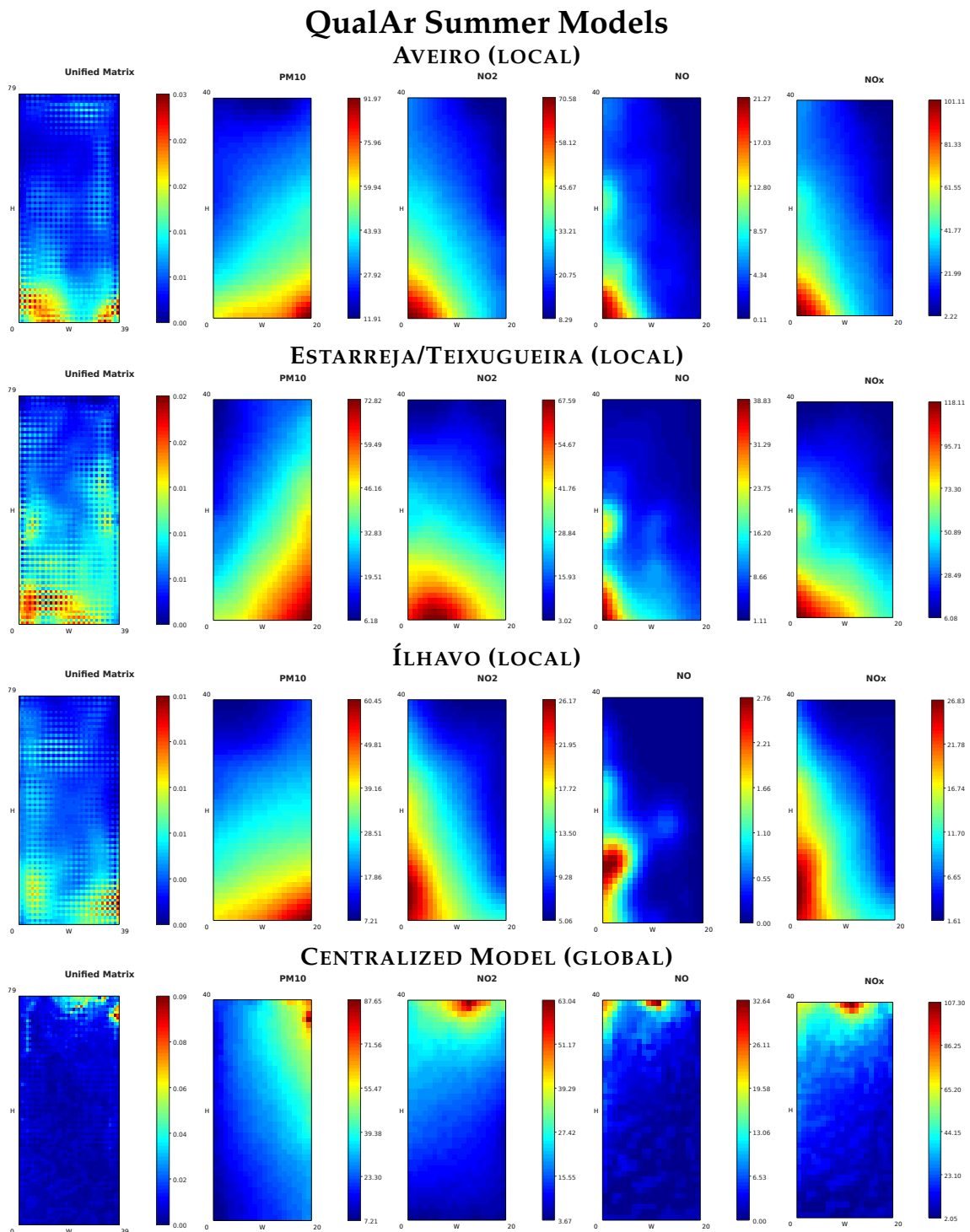
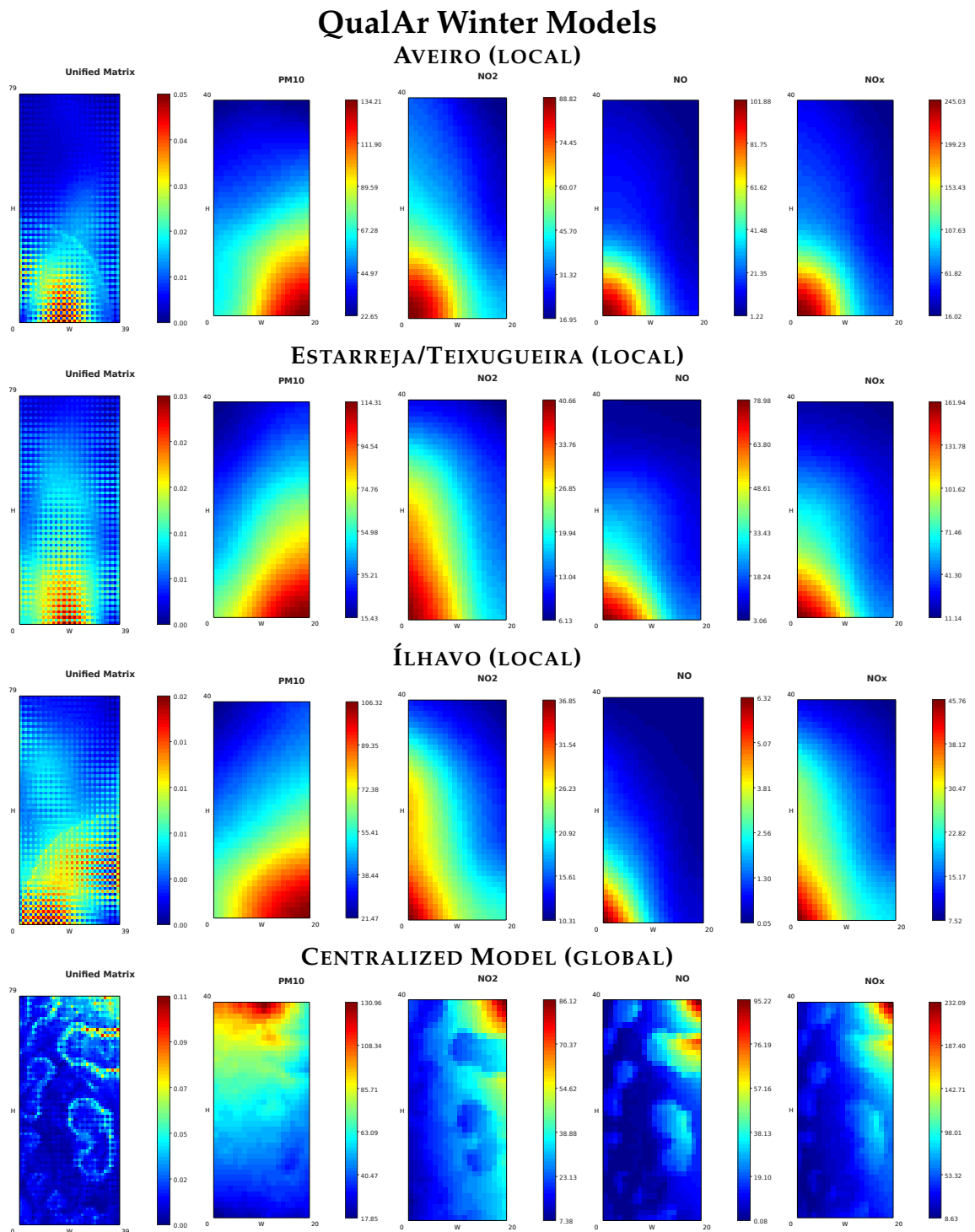


Figure 7.11: *QualAr Summer* models — derived U-Matrices and component planes.



Figure 7.12: *QualAr Winter* models — derived U-Matrices and component planes.

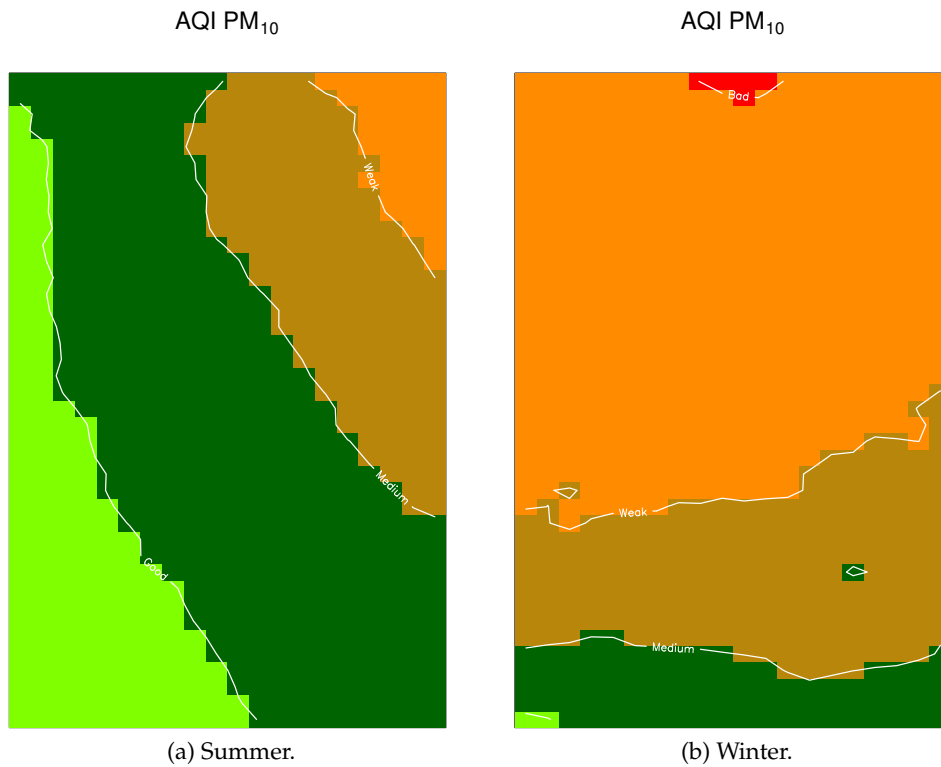


Figure 7.13: Mapping of AQI values against  $PM_{10}$  denormalized component planes of both centralized *QualAr Summer* and *Winter* models.

*Estarreja* is known for being the most polluted location in Portugal, mainly due to an industrial chemical complex that polluted the soil and river since the 50's. However, in our initial data there are no measurements related to air pollution specific to this industry, e.g., chlorine, heavy metals and organic volatile compounds, that would allow us to confirm this fact.

Contrary to the *Household* application, we will not delve into interpretation of the U-Matrices, since the type of possible knowledge inference has been demonstrated. The more interesting cluster structure concerns the centralized *QualAr Winter model*, where a complex cluster structure emerges related to the nitrogen dioxides pollution patterns, relating mainly to different concentrations between  $NO$  and  $NO_2$ .

This application to air quality monitoring data can become more interesting if we combine the information contained in the denormalized component planes with the AQI guideline values for both the obtained centralized global models. This can allow us to draw a global picture of the air quality in the district of Aveiro, in those two meteorological seasons for comparison. For example, by recovering Table 7.3, we will concentrate on the  $PM_{10}$  levels, given that the  $NO_2$  levels expressed in the respective component planes of the global models never exceed the guideline values of *Very Good*.

Figures 7.13a and 7.13b depict the centralized  $PM_{10}$  component planes for the *QualAr Summer* and *Winter models*, respectively, but mapping all feature values to the intervals

of Table 7.3, regarding concentration guidelines. By using these representative instants in the data streams we can obtain an overall indication of the average air quality in the district of Aveiro in the summer and winter. Previous pollution patterns presented until the day of the *Summer* model indicate that the average air quality is  $\approx 2/3$  in *{Very Good, Good}* indices, but otherwise in *{Medium, Weak}*. The scenario becomes worst in the winter model, where the average air quality is dominated by the *{Medium, Weak}* indices and a small expression of *Bad*. Given these results, two conclusions can be made: first, the AQI is worst in the winter than in the summer; second, and more importantly, there is a severe air quality problem in the district of Aveiro regarding  $PM_{10}$  levels during winter<sup>1</sup>.

Such AQI mappings seem interesting in a real-world application, allowing experts or the population to get a visual grasp on the average air quality pollution over time. Similar conclusions could be obtained from, e.g., *histograms*. However, generating histograms from data streams is a problematic of its own (Gama 2010) and the exemplified mappings over UbiSOM component planes can be seen as an alternative.

Applications of SOMs over air quality data can be found in literature. From this type of data, and applying the data visualization capabilities, authors in (Neme and Hernández 2011) analyzed the air quality in Mexico City and were able to detect some hidden patterns regarding the pollutant concentration, as well as to study the evolution of air quality from 2003 to 2010. Similarly, in another study, authors used the SOM to summarize ambient air quality as a collection of day types (clusters). Such clusters allowed them to identify the types of multi-pollutant combinations that occur, their temporality, and for summarizing external variables of interest, e.g., humidity, barometric pressure, wind speed. They conclude the SOM “to be an attractive framework for developing ambient air quality classification because the approach eases interpretation of results by allowing users to visualize classifications on an organized map.” — in (Pearce et al. 2014). Both these works use the SOM in a traditional *static data* setting, while this research transposes these capabilities to a possible streaming setting, where models can be inspected in real-time.

## 7.5 Feature Clustering in Financial Data

In the previous illustrated applications, the general feature clustering procedure for SOM models described in Section 5.6 can be used to additionally group correlated features, e.g., individual measurements. In this sort of applications, if the number of features is low, as was the case, we can visually detect these relationships without much difficulty. However, as the data dimensionality increases, the task becomes harder. Therefore, the feature

<sup>1</sup>Coincidentally, a news article was found after this application where the European Union warned Portugal that “The citizens of some districts of Lisbon, Porto, Aveiro, Ilhavo and Estarreja have been continuously or almost continuously exposed to harmful levels of  $PM_{10}$  since 2005, based on the most recent reports for 2012” — translated from <http://www.jb.pt/2014/09/poluicao-do-ar-em-aveiro-ilhavo-e-estarreja-leva-comissao-europeia-a-advertir-portugal/>.

clustering procedure can also be used for the purpose of grouping features or ordering their presentation to the user, as in (Vesanto and Ahola 1999).

Yet, the exemplified application in this section goes towards the fact that feature clustering in data streams is closely related to time series clustering (see Section 2.6.2). Here the set of values of a particular feature along time is regarded as an individual time series. In these sort of applications we are not interested in performing traditional clustering of observations along time, but to find (and make available at any time  $t$ ) a partition of those time series, where time series in the same cluster tend to be more alike than others in different clusters.

To this extent such an application is presented over financial data and uses the StreamART2A/SOM methodology described in Chapter 4. While the main intent was to also demonstrate the practical use of StreamART2A/SOM, this methodology has the added benefit of allowing selection of particular time intervals of the data stream (if the codebook allows) to generate the offline Batch SOM models; this contrasts with the UbiSOM in the sense that the model adapts itself continuously based on changes in the underlying data stream.

Given the feature clustering methodology uses hierarchical clustering over component planes (with results in a *dendrogram* format), it allows to observe which time series correlate more within the obtained SOM model, besides obtaining groupings by a *cuttree* algorithm, as will be demonstrated. The type of results obtained can latter be used in, e.g., in portfolio selection applications, as was proposed in (Silva and Marques 2010a).

**Disclaimer** The results contained in this section should not be used as investment advice nor there is the intent to provide any meaningful financial analysis of the data used, as it would required expert domain knowledge that is out of scope in this thesis. Therefore, the experimental results are interpreted exclusively between the original data and the various data visualizations and results obtained.

### 7.5.1 Data Description

Data was made available to this research by *GoBusiness Finance* (GoBusiness Finance 2016), an “international project focused on the research of non-normal behaviors in financial markets and the development of tools to handle with unexpected events” — is not publicly available, but was also used in (Tiple, Cavique, and Marques 2016). The original data describes stock prices collected every 2 minutes from 24 European financial institutions (see Table 7.6). Stock prices were obtained from 2015-08-07 until 2016-07-26 during open market hours, i.e., from 8:00 to 16:30 (European central time).

For generation of the *Financial data stream* date-time related features were discarded and missing values replaced by the last previous available value (feature-wise), i.e., by *last observation carried forward* method. The data stream consists in a total of 62 417 observations, with  $d = 24$ . Each observation of the data stream describes individual trading

Financial Institutions	
BNP PARIBAS	BARCLAYS PLC
SOCIETE GENERALE SA	BANCO POPULAR ESPANOL
BANCO SANTANDER SA	KBC GROEP NV
ING GROEP NV-CVA	BANCO DE SABADELL SA
BANCO BILBAO VIZCAYA ARGENTA	NATIXIS
CREDIT AGRICOLE SA	ROYAL BANK OF SCOTLAND GROUP
INTESA SANPAOLO	UBS GROUP AG-REG
CAIXABANK S.A	CREDIT SUISSE GROUP AG-REG
UNICREDIT SPA	NORDEA BANK AB
HSBC HOLDINGS PLC	UBI BANCA SCPA
LLOYDS BANKING GROUP PLC	SKANDINAVISKA ENSKILDA BAN-A
STANDARD CHARTERED PLC	INVESTOR AB-B SHS

Table 7.6: List of financial institutions (stock prices) contained in the *Financial data stream*.

minutes; each feature describes the stock price for a particular financial product. Hence the consecutive values of each feature can be regarded as a time series.

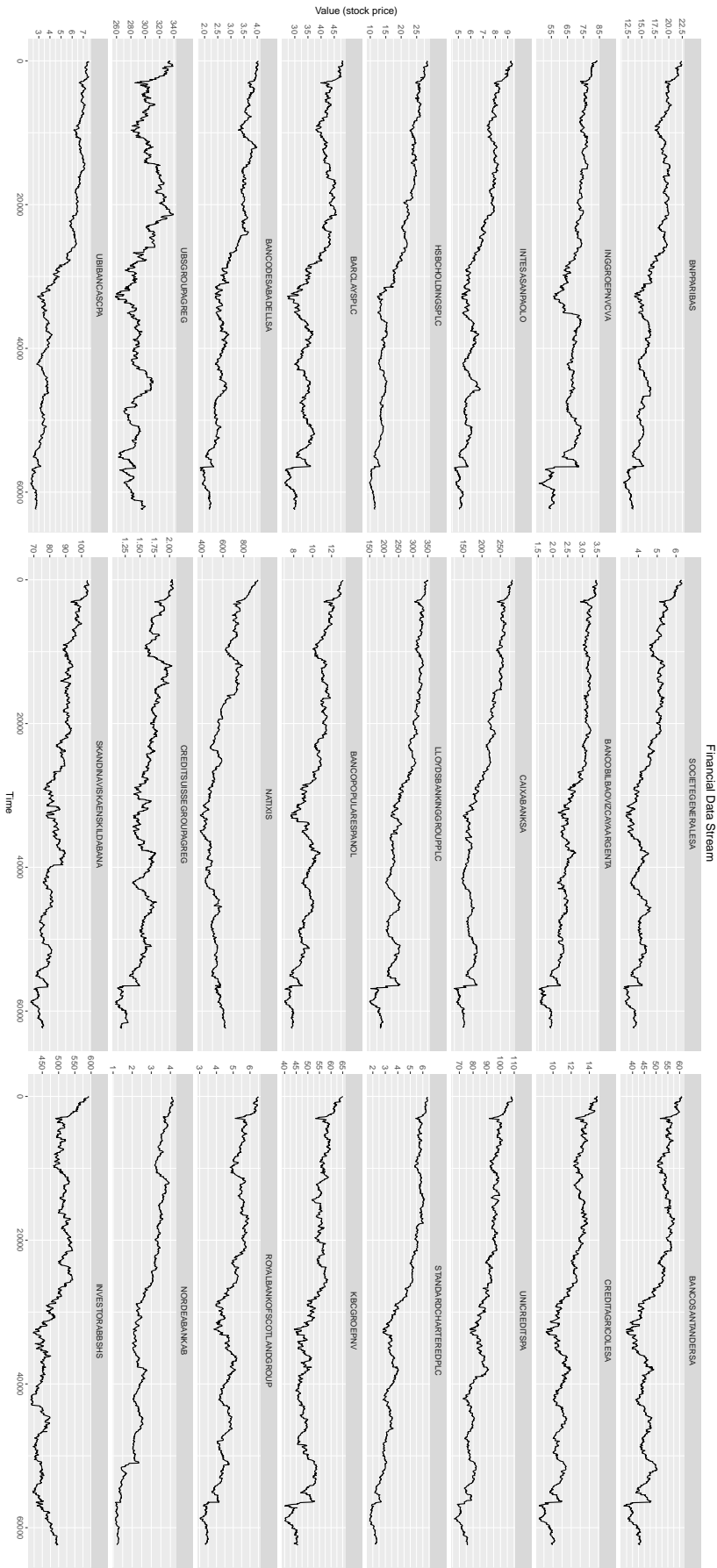
Figure 7.14 depicts the *Financial data stream* where each plot corresponds to the consecutive stock prices for the respective financial institution (feature). Some aspects should be highlighted here: *i.*) feature names are presented without any spaces or other characters between words<sup>2</sup>, and; *ii.*) only 5000 values were used to plot each feature, hence some details may not be visible. Regarding the individual time series, although global trends are similar across most features along time, there are groups of features that behave more similarly than others in shorter time frames. This is the type of groupings we intend to extract along time from this application.

All observations were normalized in the unit hypercube prior to their presentation to the StreamART2A/SOM methodology, i.e.,  $\mathbf{x}(t) \in [0, 1]^d$ .

### 7.5.2 Parameterization

The StreamART2A algorithm was parameterized with the recommended values obtained from the PSA of Section 4.5.3, i.e.,  $L = 1000$ ,  $q = 50$  and  $\eta = 0.05$ . The size of the codebook was bounded to  $K = 1000$ ; hence, from the previous parameterization this allows the codebook to store approximately micro-categories from the last 20 000 presented observations, i.e., the maximum time horizon. The *offline* Batch SOM models are obtained using a  $20 \times 40$  lattice and parameters  $\eta_i = 0.1$ ,  $\eta_f = 0.01$ ,  $\sigma_i = 1/2\sqrt{20^2 + 40^2}$  and  $\sigma_f = 1$ , with training enduring 10 and 40 ordering and convergence epochs, respectively — these values are consistent with the ones used in the experimental results of Section 4.5.5. Finally, the *average quantization error*  $\overline{qe}(t)$  assessment metric was obtained using  $T = 500$ .

<sup>2</sup>This relates to software limitations.



### 7.5.3 Application Results

Micro-categories from the StreamART2A codebook were extracted at specific time intervals and corresponding *offline* Batch SOM models generated. Given the used parameterization, namely  $K = 1000$ , the codebook can store micro-categories for a time horizon of 20 000 observations (as discussed in Section 4.3.3). All available micro-categories at a specific time  $t$  were used to generate the models. Hence, a model requested at time  $t$  describes the data stream in the interval  $\approx [t - 20\,000, t]$ . The following models, referred hereafter as *financial snapshots*, were generated along time:

**Financial Snapshot 1** requested at  $t = 20\,000$  (corresponds to the date 2015-11-25 13:32:00).

As an example this snapshot regards the time interval  $t = [0, 20\,000]$ , i.e., since 2015-08-07 08:00:00;

**Financial Snapshot 2** requested at  $t = 40\,000$  (corresponds to the date 2016-03-24 12:08:00)

and regards a time interval immediately after the previous snapshot;

**Financial Snapshot 3** requested at  $t = 62\,417$  (corresponds to the date 2016-07-26 16:30:00)

and coincides with the end of the data stream.

These points in time were selected so as to obtain three snapshots corresponding to approximately three periods of equal length during the data stream.

The component planes of each snapshot are depicted in Figures 7.15, 7.16 and 7.17, respectively. Within each figure the ordering of the component planes is the same as in Figure 7.14, line-wise. The component planes are provided for reference and also to highlight the difficulty of obtaining an overall understanding of all detected correlations between a high number of features.

Pertaining feature clustering results, i.e., time series clustering, for each of the above *financial snapshots*, the procedure described in Section 5.6 was applied using the inner product distance function between component planes. Also, the *cutree* algorithm was applied to the obtained *dendrograms* using  $k = 5$  — this algorithm partitions the features in such a way that exactly  $k$  edges of the *dendrogram* are intersected. This value was chosen arbitrarily just to illustrate the type of groupings provided. In a particular application, e.g., portfolio selection (Silva and Marques 2010a), this value can be chosen as to partition the financial products in the desired number of groups. Results for all snapshots can be found in Figure 7.18. Across all the *dendrograms*, features (time series) merged/grouped first are considered the most similar regarding their behavior in the given snapshot; then existing groupings are consecutively merged, all based on the computed distances. Hence the clusters obtained may not contain only very similar time series, but the final clusters obtained contain the most dissimilar groupings. These groupings follow the similarities between the previous component planes of Figures 7.15, 7.16







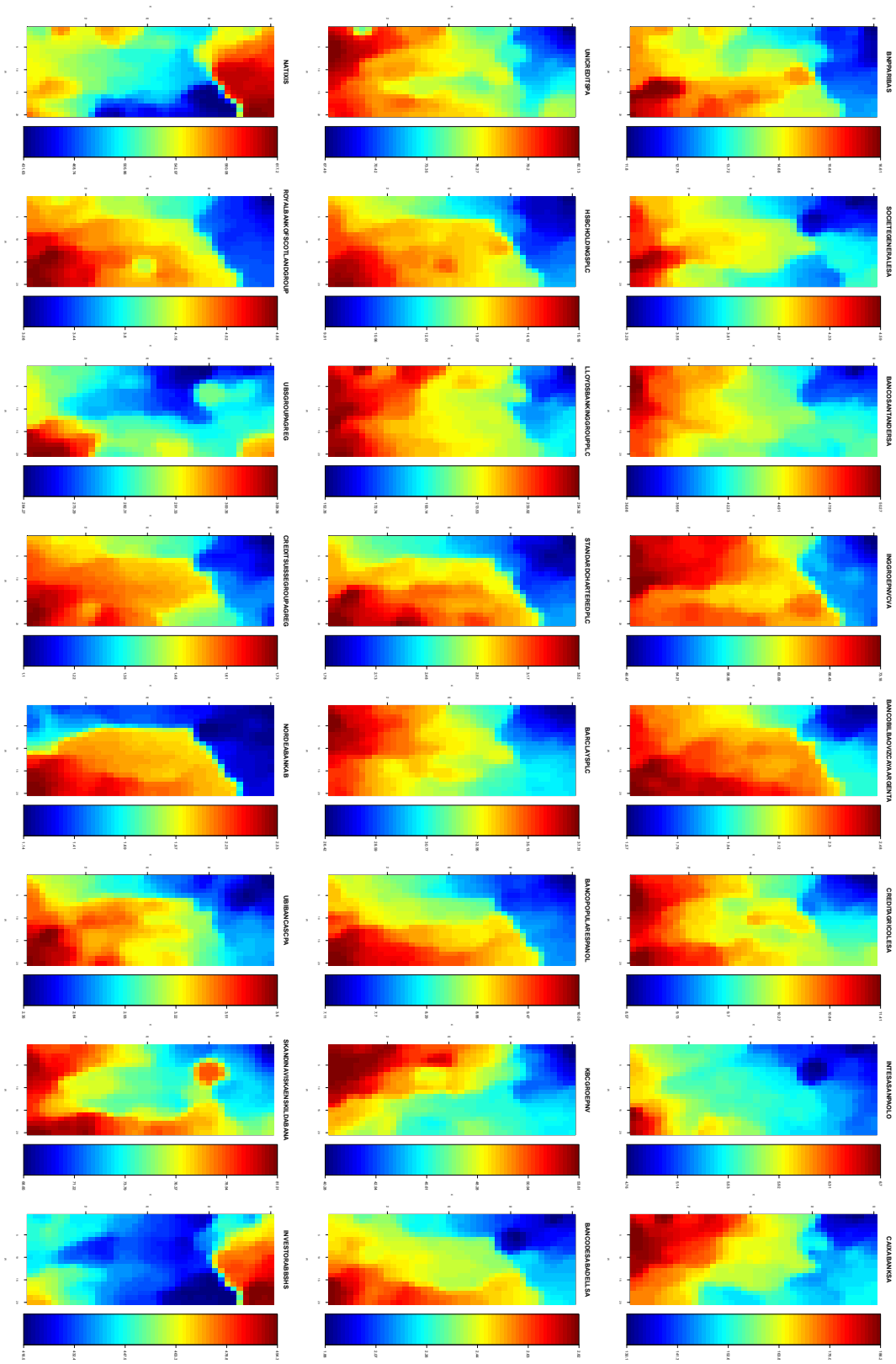


Figure 7.17: Component planes of Financial Snapshot 3.

and 7.17. Although, with such a high number of features, the comparison or validation is not visually easy, in Section 5.6 an example was presented with artificial data with a smaller number, i.e., data dimensionality.

By comparing the clustering results with the original time series (Figure 7.14) we can observe that, e.g.:

- In *Financial Snapshot 1* (Figure 7.18a), i.e., regarding  $t = [0, 20\,000]$ , *UBS GROUP AG-REG* and *INVESTOR AB-B SHS* form the pair that is merged later in the procedure; in this particular case they are clustered together. This is an example of the previous discussion where both are not comparatively very similar, but in conjunction are more dissimilar compared to all the others;
- In *Financial Snapshot 2* (Figure 7.18b), i.e., regarding  $t = [20\,000, 40\,000]$ , *CREDIT SUISSE GROUP AG-REG* is found to be more dissimilar in respect to all the others. By inspecting the data stream during this period we can observe that, e.g., approximately at the middle of this period there is an increase in the stock price not found in the other stock prices;
- In *Financial Snapshot 3* (Figure 7.18c), i.e., regarding  $t = [42\,617, 62\,417]$  we can see that, e.g., *NATIXIS* and *INVESTOR AB-B SHS* are clustered together and grouped relatively early; by comparison with the plotted time series we can indeed confirm that these two institutions have a distinct upwards trend towards the end of the data stream, which is not replicated across other institutions;
- Regarding common features clustered together between snapshots, we can find, e.g.:
  - *CAIXA BANK SA*, *UBI BANCA SCPA* and *NATIXIS* in *Snapshot 1* and *Snapshot 2*;
  - The later *NATIXIS* is clustered together with *INVESTOR AB-B SHS* in *Snapshot 2* and *Snapshot 3*. Both these findings may hint on some relation between these financial institutions.
- Overall, if we look closely, *INTESA SANPAOLO* and *SOCIETE GENERALE SA* are consistently grouped together in all snapshots, although later in the last snapshot, indicating that their correlation weakened in the last part of the data stream. This can be more easily observed when comparing the *dendrograms* side by side, as will be illustrated next.

This type of results can be used to generate and maintain investment portfolios along time, by picking assets from individual clusters as a means to diversify the investment; although, always with expert domain knowledge or through the development of informed automatic procedures.

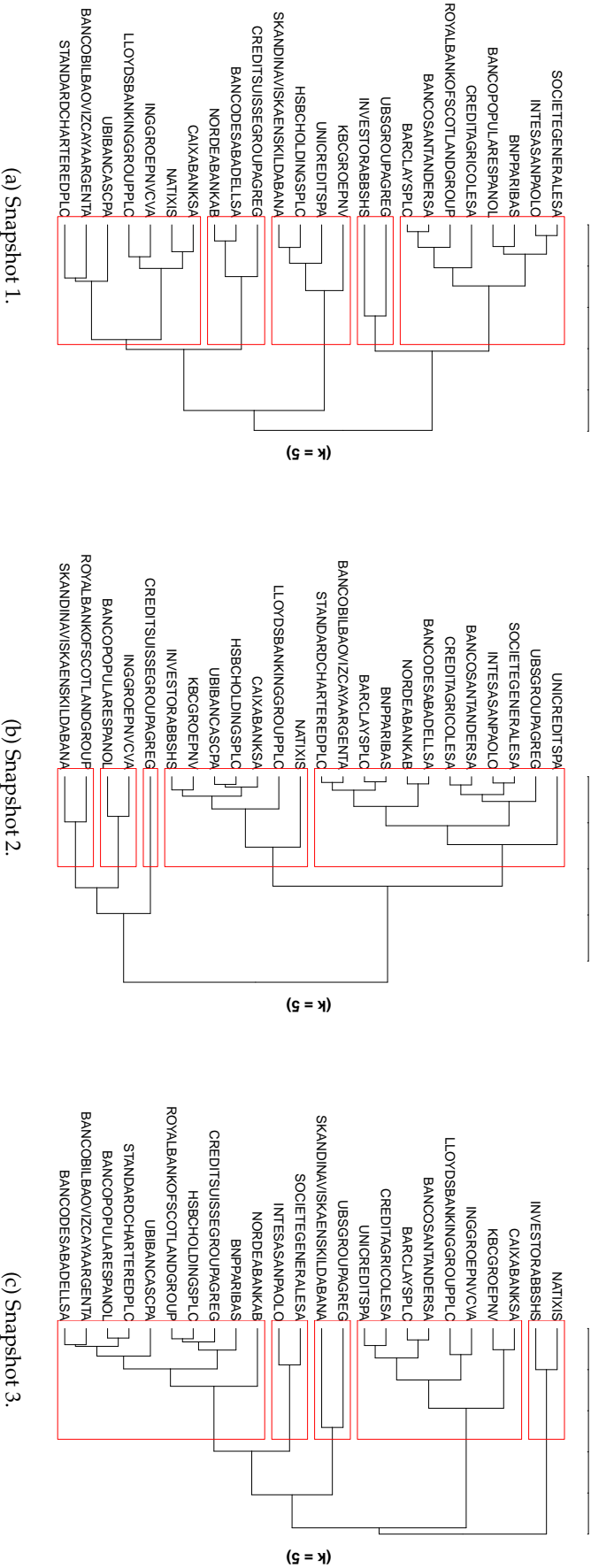


Figure 7.18: Feature clustering results for the SOM models obtained along the *Financial data stream*.

### Comparing Clustering Results Along Time

Within this sort of application it may also be interesting to understand the evolution of the similarities between the time series. Figures 7.19a and 7.19b illustrate the *entanglement* between two consecutive clustering results: connecting lines are drawn between the same features, whereas if colored means that two groupings are present in both dendrograms; dashed lines represent a combination of groupings which is not present in the other *dendrogram*. These results indicate that, e.g.,:

- Indeed, *INTESA SANPAOLO* and *SOCIETE GENERALE SA* (colored pink) maintain their close similarity along time, although less in the later snapshot (merge distance is comparatively higher);
- Between the first two snapshots, besides the pair above, we can also find that:
  - The similarities between *STANDARD CHARTERED PLC* and *BANCO BILBAO VIZCAYA ARGENTA* (colored blue) were strengthened to the end of the second time interval described by *Snapshot 2*;
  - With a similar reasoning, the same seems to have happened between *NORDEA BANK AB* and *BANCO DE SABADELL SA* (colored green);
- Finally, the overall groupings change over time (dashed lines), indicating that most similarities between time series vary throughout time;

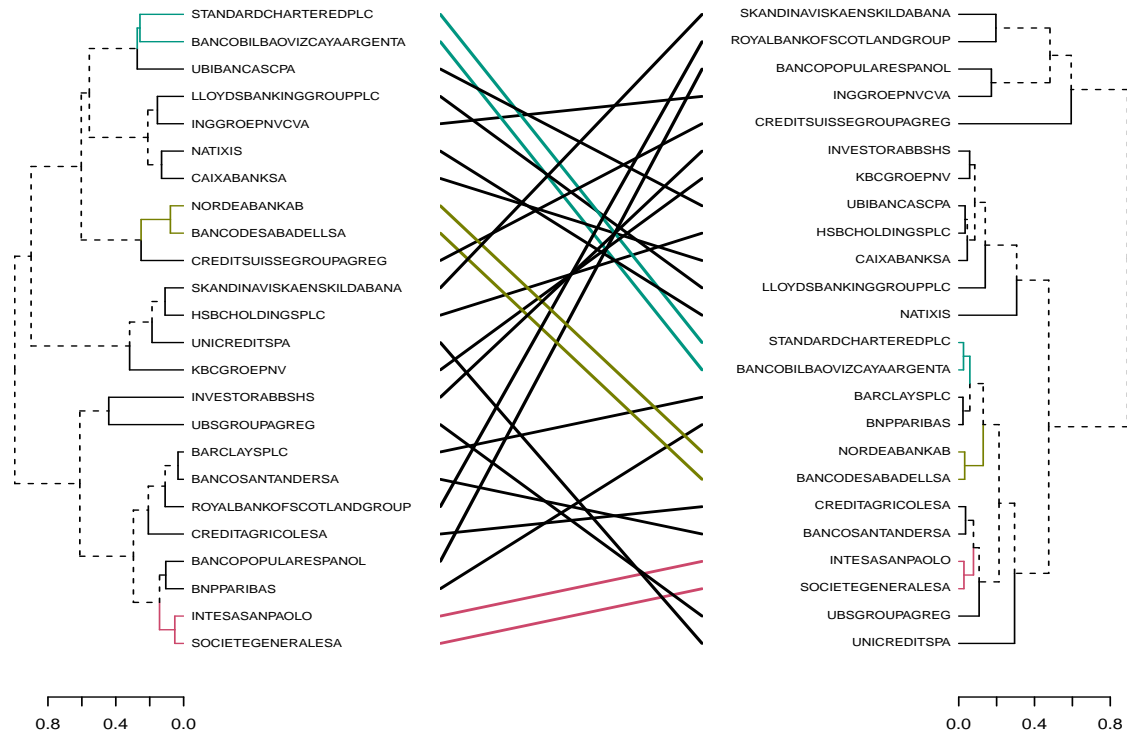
These different type of results seem to be interesting in allowing an expert to continuously monitor the correlations between stock prices and the development of automatic recommendation systems supported by this information.

Overall, in practical applications snapshots can be generated at regular intervals to perform the kind of knowledge discovery exemplified in this section.

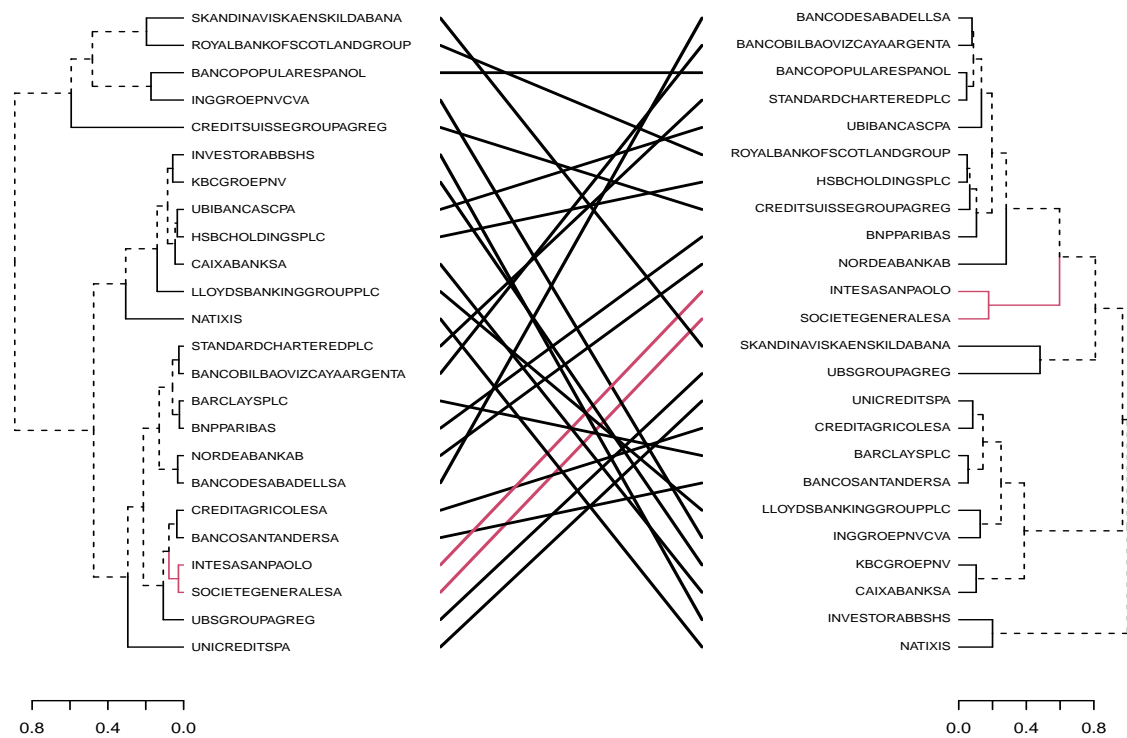
### Model Assessment and Change Detection

Finally, the *average quantization errors* along the *Financial data stream*, regarding the evolving StreamART2A codebook, are depicted in Figure 7.20. We can observe constant variations of the metric along the data stream, but with a pronounced change just before  $t = 60\,000$ . This event is within the time frame of the *Brexit* poll occurred at 2016-06-23 and may explain such change.

As mentioned in Section 4.6, an automatic change detection mechanism, involving the comparison of two *average quantization errors* of different lengths, was proposed in (Silva, Marques, and Panosso 2012) and illustrated with other financial data, i.e., stock market indexes and computed statistics. However, the determination of the window lengths was achieved through external expert knowledge for that particular data.



(a) Financial Snapshot 1 and Financial Snapshot 2.



(b) Financial Snapshot 2 and Financial Snapshot 3.

Figure 7.19: Entanglement between consecutive financial clustering results, obtained from the *Financial Snapshots*.

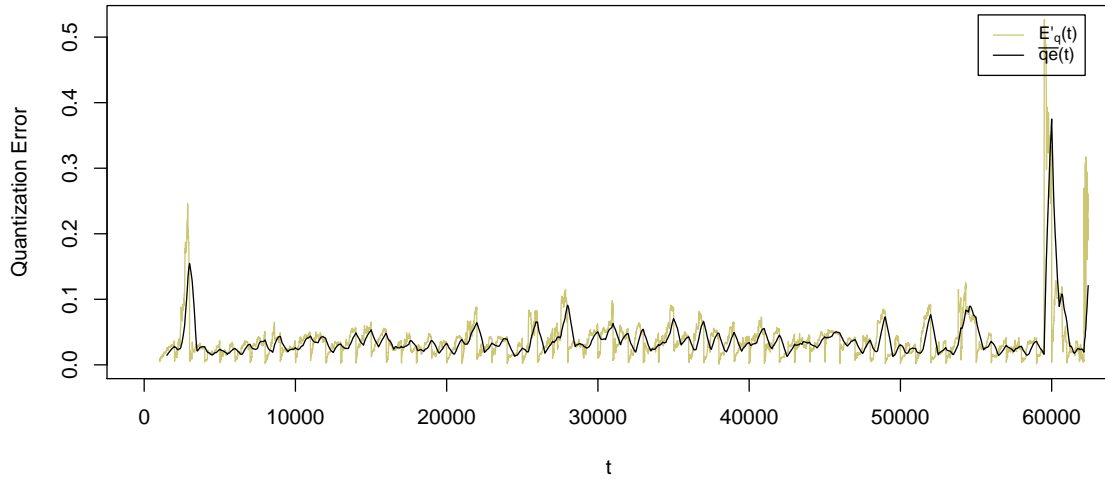


Figure 7.20: Average quantization errors for the Financial data stream.

## 7.6 Remarks and Future Work

In this chapter applications of the StreamART2A/SOM methodology and the UbiSOM algorithm and related methodologies were applied to real-world data, as a means to exemplify its usefulness in the respective real-world scenarios. Monitoring applications for sensor data (electrical consumption and air pollutant patterns) and clustering financial time-series were presented, while exemplifying the versatility of the obtained SOM models in performing clustering of observations and clustering of features, i.e., time-series.

All the exemplified applications, and similar ones, can be deployed in the real-world with the developed software presented in Appendix A, namely the contributed framework. For example, the Household application can be deployed easily if one is able to obtain individual measurements of appliances and/or divisions in the household. An interesting recent development in Portugal, regarding the feasibility of such approach, is the *Energias de Portugal (EDP) RE:DY*<sup>3</sup> system (*Eletricidade de Portugal* 2016), where household appliances can be connected to a *smart box* for remote querying and control. One can envision the consumption streams being fed into the contributed framework for continuous real-time monitoring and clustering of consumption patterns, as exemplified. The framework also allows to perform feature clustering either through the remote web interface or via *R* integration. The later also permits more advanced manipulations of the obtained SOM models, as exemplified in the AQI overlap of Figure 7.13 and the comparison of consecutive clustering results of Figure 7.19. Moreover, the standalone UbiSOM library allows developing, e.g., interactive data visualization techniques for financial data (Marques, Silva, and Santos 2016).

<sup>3</sup>Stands for “Remote Energy DYNamics”.

Future work regarding the illustrated applications is discussed in the following paragraphs.

**Data normalization.** This aspect was discussed in Section 3.5, putting forward the assumption that data is normalized when using the proposed methods and algorithms. Quickly recapping, normalization of data streams requires that lower and upper bounds of individual features values are known. In sensor data, for example, data is bounded to the range of the sensor outputs themselves. In financial data, one can establish bounds from historical information or within a window model (Ogasawara et al. 2010). Nonetheless, application of strategies to normalize data streams was considered beyond the scope of this thesis.

However, data stream normalization is critical aspect that should be addressed in future work so as to improve the practical deployment of the proposed methods.

**Missing data.** We should be aware that real-world data, specially obtained in real-time is prone to missing values. These may arise in sensor data either due to sensor failure or maintenance; also in financial data collecting errors may occur; an extreme case is if the data stream is composed by financial products from different stock markets located in different countries, where different opening-closing hours and local holidays may determine the absence of values for particular stocks. However, applying clustering methods to financial data obtained from different stock markets across the globe should be approached differently, e.g., by generating models for individual stock markets and then combining the results; however this needs a completely novel approach regarding the proposed distributed learning methodology, because the individual problems are different, i.e., address different features.

In some applications presented in this chapter all observations with missing data were filtered out. This may be unrealistic in practice, but several options exist, e.g., i.) in the simplest cases a pre-processing engine can fill the missing values with the last previously known, or; ii.) contemplating missing values within the algorithms. Relatively to the later approach, e.g., several works have addressed the SOM and missing data problematic. In most of the SOM literature the missing values are treated as in (Samad and Harp 1992), e.g., the best-matching units for the observations with missing values are computed by omitting the individual missing values in the distance metric; the missing values are ignored also while updating the prototype vectors. This approach is implemented in the widely used Matlab's SOM Toolbox (Vesanto et al. 1999) and has been applied to many kinds of data, such as socioeconomic data (Cottrell and Letrémy 2005), industrial data (Rustum and Adeloje 2007) and financial data (Sorjamaa et al. 2009). Given this, somewhat, established manner of dealing with missing data, its extension to the StreamART2A and UbiSOM algorithms seems trivial in the future.



**Noisy data and outliers.** Noisy observations also occur in real-world data, and sensor data is a typical example. Sensors can be subject to electromagnetic interference or hostile measurement environment, for example. Regarding noise, the SOM is known for its robustness in dealing with noisy data, but can be affected by extreme outliers in data (Allende et al. 2004). By extension, UbiSOM inherits the robustness to noise from the underlying dynamics of the prototype update model of the basic SOM (an outlier only affects the best-matching unit and its neighbors), but implicitly minimizes the outliers effects, e.g., by estimating the learning parameters by averaged metrics, namely the average quantization error, the impact of outliers is expected to be diluted in the other observations while computing the average error. Nonetheless, effective research regarding the impact of outliers in the UbiSOM algorithm should be addressed in future work. Regarding the StreamART2A/SOM methodology, this particular aspect was discussed in Section 4.6.

**Heterogeneous data.** Real-world data may also include heterogeneous data, in the sense of observations containing mixed data types. Although the StreamART2A and UbiSOM algorithms were proposed for real-valued features, there are proposals in literature to additionally process categorical data, e.g., (Chen and Marques 2005). This, however, requires that all possible categorical values are known before hand and a preprocessing stage is included to translate the categorical values to numerical ones.

**Spatial data.** Unfortunately, no application for the collaborative learning methodology was presented with real-world data. Its application seems more interesting when integrating geographic locations, i.e., spatial features, within the UbiSOM update rule, as presented and discussed in (Baão, Lobo, and Painho 2005b). This could then allow to collaboratively obtain UbiSOM models that could be projected onto geographic maps for participatory sensing applications, e.g., (Dutta et al. 2009). Therefore, integration of spatial data seems an interesting research path for future work to leverage the proposed collaborative learning methodology.





# Conclusions

Those who stand for nothing fall for anything.

---

ALEXANDER HAMILTON, POLITICIAN, ECONOMIST AND  
AUTHOR, BORN IN CHARLESTOWN, SAINT KITTS AND  
NEVIS (1755-1804)

This chapter tries to summarize the main findings, contributions, and recommendations that can be extracted from this thesis, by revisiting the research questions posed throughout the manuscript. Finally, future practical application scenarios, intended collaborations and open research issues are addressed; hopefully, they can be reference for future work.

## 8.1 Main Findings and Contributions

This thesis posed the following initial research questions:

**RQ1** *What are the limitations of Self-Organizing Maps regarding streaming data?*

**RQ2** *Can Self-Organizing Maps provide a valuable tool for data stream cluster analysis regarding current methods? If so, which research paths should be pursued in terms of relevant contributions?*

**RQ3** *How can the ubiquitous aspect of data streams be tackled with SOMs?*

Regarding research question **RQ1**, despite being a well established algorithm with hundreds of applications and dozens of variants in literature (Pöllä, Honkela, and Kohonen 2009), no prior work existed regarding the SOM and data streams — at least not clearly addressing the data stream problematic. The limitations of the standard SOM

algorithms in learning from non-stationary data streams were addressed in Section 2.4, while presenting this ANN. The main deterring aspect is the use of time-dependent annealing schemes to guide the evolution of the learning parameters and, consequently, the convergence of the maps. These cannot be applied over unbounded data streams. Based on this limitation, desire to maintain the visual data exploratory abilities of the SOM and general requirements imposed by the data stream model (Barbará 2002), proposed requirements for SOM variants addressing cluster analysis of data streams were put forward in Section 2.5.1 (these are again summarized in Section 8.1.2). A literature review was conducted for existing variants that could meet those requirements (in Section 2.5.2) and no existing variant was deemed fit for the data stream problematic in the light of the above requirements, mainly due to irregular network topologies, e.g., ESOM (Deng and Kasabov 2003) and SOINN (Furao and Hasegawa 2006), deterring the use of the visualization procedures, or poor quantization performance of the input space, e.g., PLSOM (Berglund 2010) and DSOM (Rougier and Boniface 2011), which may ultimately also affect the visual knowledge discovery process. Consequently, room for improvement regarding methods and algorithms that effectively allow the use of SOMs for exploratory cluster analysis over data streams was identified at this point.

In answer to research question **RQ2**, a survey of current methods for cluster analysis over data streams was presented in Section 2.6, which served two purposes: first, to confirm the research gap in the dedicated literature regarding the use of SOMs; second, to review current established methodologies in dealing with data streams, namely for cluster analysis. It was found that current methods involve adaptations of popular clustering algorithms to stream settings, e.g., k-means in *Single-pass k-means* (Farnstrom, Lewis, and Elkan 2000) or, more commonly, where a two-phase approach is followed. This strategy focuses on *online*, efficient and evolving data stream abstraction (summaries) structures, from where *offline* cluster results are obtained on demand by applying traditional clustering algorithms over the current summaries. Therefore, following the taxonomy in (Han, Kamber, and Pei 2006) for clustering algorithms, several current proposals were identified, e.g.:

- *partitioning*: CluStream (Aggarwal et al. 2003);
- *hierarchical*: BIRCH (Zhang, Ramakrishnan, and Livny 1996);
- *density-based and grid-based*: D-Stream (Chen and Tu 2007);
- *model-based*: SWEM (Dang et al. 2009).

Also, the majority of existing proposals (including the above) address the problem of clustering observations, whereas literature on feature clustering algorithms is seldom, e.g., ODAC (Rodrigues, Gama, and Pedroso 2008). Also found, is that feature clustering in a stream setting is intimately related to time series clustering (Gama 2010).

Within the previous taxonomy, the SOM can be characterized as a model-based algorithm and its clustering results were compared to those of other traditional algorithms, that are used in current proposals, in Section 2.4.7. All combined, its visualizations (even understandable by non-experts), no prior assumptions on data, description of clusters and versatility in performing either clustering of observations and features were found to be highly attractive regarding current methods. Hence, the extrapolation of these capabilities to a stream setting, favoring visual data exploration, was considered of great interest and to indeed fill a gap in the current panorama of methods (see ).

Attending to the fact that most established methodologies involve two-phase approaches to data streams, an initial established goal was to mimic such approach so as to obtain *offline* SOM models. To this extent, ART networks were found to be an interesting approach for the *online* abstraction procedure. Nonetheless, a SOM variant tailored for data streams was considered a fundamental contribution. Hence, two additional research questions were formalized and later answered, through contributions made in this thesis:

- *Can ART networks produce an efficient data stream summary from where SOM models can be generated?*
- *Can Self-Organizing Maps learn non-stationary data streams, while keeping the original SOM properties intact?*

This thesis also targeted the ubiquitous aspect of data streams, i.e., its distributed nature in several real-world application scenarios, e.g., sensor networks (Gama, Rodrigues, and Lopes 2011) and participatory sensing (Dutta et al. 2009). Hence, regarding research question **RQ3**, from Section 2.6.3, the most promising strategy was found to focus on the trade-off between *local* and *global* models, i.e., instead of centralizing all data streams to produce clustering results (which is unfeasible in respect to scalability), this strategy concerns having each location/device maintain a local compact model describing the current underlying distribution of the local data stream; then, only these models (data summaries) need to be centralized or shared between devices to obtain the global models and, therefore, global clustering results. Consequently, the additional research question raised here was:

- *Can distributed and collaborative learning strategies over ubiquitous data streams be devised using Self-Organizing Maps?*

With a practical aspect in mind, the aggregating question for all the above research question regarded concrete real-world applications that could benefit from the undergone research. Therefore, the last formalized additional research question was:

- *What sort of real-world problems can benefit from this research?*

### 8.1.1 Summary of Contributions

Directed mainly towards the above additional research questions, this thesis contributed several methods and algorithms (supported by other minor contributions and software) and can be summarized as follows:

- i. Establishment of requirements SOM variants should target when addressing data streams;
- ii. The *StreamART2A* algorithm for abstracting an incoming data stream, from where offline SOM models are generated for a specific time-horizon. This methodology is referred to as the *StreamART2A/SOM* approach.
  - (a) a constrained variant of the ART2-A algorithm and the micro-category concept;
  - (b) a modified update rule for the Batch SOM algorithm to learn from micro-categories;
  - (c) proposal of the *average quantization error* metric;
  - (d) parameter sensitivity analysis of the StreamART2A algorithm and general evaluation with artificial data.
- iii. The *Ubiquitous Self-Organizing Map* (UbiSOM) algorithm, a SOM variant tailored for non-stationary data streams;
  - (a) inclusion of temporal information within each UbiSOM neuron;
  - (b) proposal of the *average neuron activity* metric;
  - (c) proposal of a weighted drift function to estimate learning parameters in a time-independent fashion;
  - (d) parameter sensitivity analysis of the algorithm and general evaluation with artificial data;
  - (e) proposal of an automatic feature clustering method for SOM models; evaluated with UbiSOM for artificial data.
- iv. Methodologies that explore the UbiSOM algorithm in ubiquitous environments leveraging its capabilities to produce local models that can be centralized and/or shared into other global UbiSOM models.
  - (a) a codebook filtering mechanism to remove unrepresentative prototypes;
  - (b) a codebook merging mechanism to remove duplicate prototypes;
  - (c) proposal of *distributed* and *collaborative learning* methodologies targeting the trade-off between local and global models;
  - (d) evaluation of the proposals with artificial data.

- v. A set of real-world applications for contributions in *ii.*), *iii.*) and *iv.*).
  - (a) application scenario of household electric data and clustering of consumption patterns using the UbiSOM; proposal of two new component plane-like visualizations for the UbiSOM temporal data extensions.
  - (b) application scenario for a distributed air quality monitoring network, using the UbiSOM and *distributed learning* strategy ;
  - (c) application of the StreamART2A/SOM methodology to financial data, by clustering financial time series along time and presenting the evolution of the correlations.
- vi. Contributed Software and general purpose library for the UbiSOM algorithm and related methodologies.
  - (a) an UbiSOM framework that allows the deployment of the algorithm over an input data stream as a service that can be remotely queried;
  - (b) a general purpose library to develop other applications.

While contributions *ii.*) and *iii.*) are different strategies to address single multi-dimensional data streams, contributions *iv.*) were proposed only for the UbiSOM, i.e., ubiquitous environments. Also, the contributed software at this point only contemplated the UbiSOM. Nonetheless, real-world application scenarios were exemplified for all proposed methodologies in contribution *v.*), except the *collaborative learning* strategy of *iv.*), as discussed in Section 7.6.

The following sections detail the above contributions and provide answers to respective additional research questions that they may target.

### 8.1.2 Requirements for SOM Variants in Dealing with Data Streams

Derived from **RQ1**, requirements for SOM variants tailored for data streams were initially addressed in (Silva and Marques 2013) and contributed, namely: fixed topology, i.e., a regular lattice of neurons to allow visualization procedures; time-independent learning parameter estimation; proper convergence to the underlying distribution in the sense of a VQ procedure; incremental and efficient processing of individual observations; compactness of the model; dealing with evolving data (non-stationarity); robustness to noise, and; handling high-dimensional data.

While the first three requirements are SOM related, the others are general to any data stream clustering algorithm. These requirements were all fulfilled by the UbiSOM algorithm and, consequently, based on experimental evidence, are found to be supported in theory and in practice.

### 8.1.3 A Two-Phase ART/SOM Approach to Data Streams

**Research question** *Can ART networks produce an efficient data stream summary from where SOM models can be generated?*

**Methodology** In Chapter 4, inspired by the popular two-phase approach to data streams, a similar one was proposed towards using the SOM to perform offline exploratory cluster analysis (Silva and Marques 2012). The data stream abstraction procedure is performed by a constrained ART2-A algorithm that dynamically adjusts the vigilance  $\rho$  to produce at most  $q$  prototypes from  $L$  observations. This constrained algorithm and resulting codebook is called StreamART2A and introduces the concept of micro-category, i.e., a prototype with extended information. This later concept was proposed following the currently established *cluster feature* and *micro-cluster* structures. The StreamART2A algorithm is responsible for processing an incoming data stream through a landmark window of size  $L$ , from where consecutive sets of  $q \ll L$  micro-categories are produced; the total number of micro-categories kept, i.e., the codebook, is managed by a FIFO structure and bounded by  $K$ , therefore only describing the most recent observations and dealing implicitly with change. Within a maximum time horizon permitted by the current model, a set of micro-categories can be recovered and used to generate an offline Batch SOM model to perform exploratory clustering — the embedded information within the micro-categories is taken into account by a modified update rule. The *average quantization error* ( $\overline{qe}(t)$ ) — a moving average of quantization errors computed over a sliding window of length  $T$ , was also proposed as a means to monitor the evolution of the fit between the current codebook and the evolving data stream (Silva and Marques 2012; Silva, Marques, and Panosso 2012).

**Findings** The granularity of the StreamART2A summarization should be adequate so as to capture the underlying distribution and to produce meaningful SOM models, i.e., a minimum number of available micro-categories is required for a particular SOM lattice size. This granularity is dependent of the StreamART2A parameterization. Through experimental PSA a good set of overall parameters for the StreamART2A algorithm was found, namely  $L = 1000$ ,  $q = 50$  and  $\eta = 0.05$ . Exploratory cluster analysis over artificial data streams matched the expected outcomes. Also, the application of the methodology to real-world financial data, namely clustering of time series, provided evidence of its practical applicability and, altogether, versatility in allowing clustering of observations and features. The empirical validation of the  $\overline{qe}(t)$  metric showed that it can indeed be used to assess the convergence of a codebook to an evolving data stream, although not being consistent in respect to error increase in the presence of all sorts of change, e.g., the error decreases with the disappearance of clusters. This is due to the fact that in the subsequent landmark windows there is more micro-categories to describe a lower number of clusters.



**Assessment** Overall, the experimental results provide strong evidence towards the viability of the approach and support the use of ART networks as a means to produce online summarizations that can be later leveraged by the SOM. However, as with any two-phase approach, clustering results are delayed by the choice of the time interval from which to recover micro-categories and respective offline model generation. Hence, this approach may not be the best solution for real-time monitoring applications, which motivated the UbiSOM algorithm.

#### 8.1.4 The Ubiquitous Self-Organizing Map

**Research question** *Can Self-Organizing Maps learn non-stationary data streams, while keeping the original SOM properties intact?*

**Methodology** In Chapter 5, the UbiSOM algorithm was presented as a novel SOM variant tailored for non-stationary data streams (Silva and Marques 2015a; Silva and Marques 2015b). The UbiSOM is able to properly converge in the sense of a VQ procedure during stationary phases of the data stream, mimicking the original SOM, while being able to react to changes in the underlying data stream. As opposed to other fixed topology proposals that use only the local error  $E_q(t)$  to estimate learning parameters, e.g., PL-SOM and DSOM, the UbiSOM algorithm takes a different approach considering global metrics to assess convergence of the codebook to the underlying distribution of the data stream. This is achieved by estimating learning parameters through the previous *average quantization error* ( $\overline{qe}(t)$ ) and proposed *average neuron utility* ( $\overline{\lambda}(t)$ ), weighted in a drift function  $d(t)$  used to estimate learning parameters in a time-independent fashion. The  $\overline{\lambda}(t)$  metric was introduced due to the fact that the  $\overline{qe}(t)$  metric, when applied to the UbiSOM, may not denounce the disappearance of clusters. The use of *Gaussian* moving averages to effectively implement the previous assessment metrics was suggested. In the UbiSOM algorithm, the same set of prototypes are updated during processing of the data stream, as opposed to the StreamART2A algorithm where they are continuously generated. Therefore, e.g., if a cluster disappears the prototypes that were quantizing it do not contribute to any change of this metric. The algorithm also uses a finite-state machine approach, consisting in the *ordering* and *learning* states: in the former, the algorithm is able to initially unfold from a totally random initialization in a relatively short amount of time, while in the later, proceed with time-independent learning parameter estimation that is able to explicitly react to change. In case of severe changes in the underlying distribution the algorithm can transition back to the ordering state.

**Findings** From experimental PSA, using artificial data, it was found that the UbiSOM is more sensitive to the parameterization, e.g., as opposed to the StreamART2A/SOM approach. However, good overall intervals for the UbiSOM specific parameters  $\beta$  and  $T$  were found, and later confirmed with real-world data, namely  $\beta \in [0.6, 0.9]$  and  $T \in$

[1000, 2500]. The parameter values empirically established for the other parameters, namely  $\{\eta_i, \eta_f, \sigma_i, \sigma_f\}$ , performed well across all tested artificial and real-world data streams.

Evaluation of clustering results both for observations and features using artificial data produced the expected results, using the obtained parameters from the PSA. Also, concerning comparable algorithms, the UbiSOM algorithm outperformed the *Online SOM*, *PLSOM* and *DSOM* algorithms on non-stationary data streams, regarding utilized error metrics.

By observing the evolution of the learning process, i.e., assessment metrics and learning parameters, across several problems, the algorithm was found to exhibit complex dynamics due to the inherent feedback mechanisms, i.e., between the output errors used to compute the assessment metrics, which in turn are used to estimate learning parameters.

**Assessment** The algorithm fulfills the requirements established previously, but robustness to noise should be further investigated. As such, assuming a good parameterization, the algorithm is indeed able to “learn” non-stationary data streams, while keeping the original SOM properties intact, allowing real-time visual exploratory cluster analysis. Experimental evidence from artificial and real-world data also attest the viability of the algorithm and versatility in allowing clustering of observations and features. Given the novelty of the approach and results obtained, the UbiSOM algorithm can be regarded as the most interesting contribution of this thesis.

### 8.1.5 Distributed and Collaborative Learning of Ubiquitous Data Streams

**Research question** *Can distributed and collaborative learning strategies over ubiquitous data streams be devised using Self-Organizing Maps?*

**Methodology** In Chapter 6, methodologies addressing the use of the UbiSOM algorithm in ubiquitous environments were presented, as initial approaches to this problematic. Proposed methods aimed at the current trade-off between local and global models, where the bulk of the processing is put on the local nodes, e.g., sensors with computing and communication capabilities. Here, local models can be seen as subjective views of the data, while global models more general and inclusive views of the problem.

Assuming each node on the network processes its own acquired data stream (pertaining the same problem/task), continuously maintaining a local UbiSOM model, two learning strategies were tackled — *distributed* and *collaborative*. The former consists in generating a one-shot global centralized SOM model from the current codebooks of a set of distributed nodes; in this setting the Batch algorithm was used, for which, if scalability is a concern, parallel implementations are known (Silva and Marques 2007). The later assumes nodes can move in the physical space and consists in collaboration between nodes through, e.g., proximity-based opportunistic networks. The proposed collaborative methodology allows each node to maintain a subjective view of the problem through

its local model, while the global UbiSOM model evolves continuously with prototypes from different sources, namely the prototypes from its models and from other nodes it interacts with (Silva and Marques 2010b).

Leveraging the extended temporal information in the UbiSOM neurons, a method was proposed to filter unrepresentative prototypes from local UbiSOM codebooks prior to their communication. Also, another proposed method is used to remove duplicate prototypes, i.e., that occupy the same region the the data space, when merging local codebooks. Both previous learning strategies rely on these proposed techniques.

**Findings** Experimental evaluation, through a simulated ubiquitous environment using artificial data, indicated that the proposed methodologies were able to produce the expected results — models of the local nodes learning different parts of the input space were shared (between nodes) and later centralized. In the *collaborative learning* strategy, the nodes were indeed capable of collaboratively learn overall portions of the input space. In the *distributed learning* strategy, the global model was found to be identical to the model of a node learning the entire input space.

The distributed learning strategy is simple in its conception and rather straightforward. As such, an application with real-world data was achieved with success. On the other hand, the collaborative learning strategy raises additional difficulties, such as reduction of the accuracy of the global models as more codebooks are incorporated.

**Assessment** Towards the above research question, the proposed methodologies exhibited promising results, but should be further studied, namely the *collaborative learning* strategy. Also, the lack of a current automatic change detection mechanism degrades the scalability of the approach, e.g., in the distributed learning strategy only models that have changed should be sent in a subsequent request from the central location.

### 8.1.6 Real-World Applications

**Research question** *What sort of real-world problems can benefit from this research?*

**Methodology** Towards the above research question, a set of real-world application scenarios were exemplified using real-world data, for the UbiSOM algorithm and related additional methodologies, e.g., distributed learning of ubiquitous data streams, as well as for the the StreamART2A/SOM methodology and feature clustering abilities. The demonstrated application scenarios for the UbiSOM revolved around monitoring applications. From the extended temporal information contained within each UbiSOM neuron two new visualizations were introduced, namely the *BMU Map* and *Activity Map* visualizations, as a means to aid the user in understanding the recency of the modeled information; it can also provide a visual confidence mechanism in the current UbiSOM model, besides the trend of the  $d(t)$  function. These examples were presented to highlight

the real-time exploratory cluster analysis from local data streams and the generation of global SOM models from distributed local models.

Another application scenario was exemplified for the StreamART2A/SOM methodology using financial data. Here, the proposed feature clustering methodology was applied to consecutive SOM models obtained along the data stream so as to generate groupings and clusters of financial time series.

Real-world data streams were obtained from a variety of sources. Data for the *Household* application, regarding household electric consumption data, came from the UCI repository (Lichman 2013); for the *QualAr* application, regarding distributed air quality monitoring, data was obtained from the *QualAr* (QualAr website) website, and; for the *Financial* application, regarding clustering of financial time series, data was provided by *GoBusiness Finance* (GoBusiness Finance 2016). The UbiSOM parameters for the respective problems were derived from a PSA, using a subset of the data streams. The StreamART2A/SOM parameters used were the ones previously suggested.

**Findings** In the *Household* application, the UbiSOM exhibited the capability to continuously model the electric consumption patterns, as they changed over time. The application of the visualizations allowed the detection of groups of consumption patterns (clusters) and corresponding descriptions. It also suggested some correlated features. The proposed new visualizations were found to indeed be helpful regarding confidence on the models and, indeed indicated that the UbiSOM was converging properly during the depicted snapshots.

The *QualAr* application provided evidence of the viability of the distributed learning strategy over sensor network applications. Focusing on the district of Aveiro, for the monitored pollutants, higher concentrations were to occur in the winter and in the city of Aveiro. Superimposing AQI thresholds over the component planes of the centralized (global) models allowed to gain insight of the overall air quality in the district.

In both above applications, the PSA using real-world data confirmed the previously suggested parameter intervals from artificial data.

Concerning the *Financial* application, results seemed interesting in providing groupings (clusters) of financial time series that behave similarly along time. While this can be used for, e.g., portfolio selection applications, comparison of consecutive clustering results was also demonstrated, which allow to visualize the evolution of correlations between time series.

In all applications, the results were empirically compared to the data streams at the instants where the models were obtained/analyzed, providing strong evidence of their correctness.

**Assessment** While we cannot safely assume that the current proposals are applicable to any real-world problem that produces local, or ubiquitous, data streams, three specific real-world application scenarios were addressed. The type of results obtained point

towards the usefulness of the proposals, the versatility of the methods and interesting post-processing of obtained SOM models, e.g., superimposing AQI thresholds onto component planes and comparison of subsequent models.

### 8.1.7 UbiSOM Framework and Library

Directed towards application scenarios of the UbiSOM, a body of contributed software also resulted from this thesis. With the developed framework any local monitoring application, e.g., similar to the *Household* application, can be easily deployed and allows remote querying of the UbiSOM models through presentation of the standard visualizations in a web browser. The feature clustering method is also included. Additionally, an R library was developed to allow post-processing of UbiSOM models as illustrated in the *QualAr* and *Financial* applications. Therefore, the contributed software allows the deployment of the presented real-world applications and others related. Moreover, with the contributed standalone library, different applications can be developed by others. The contributed software is presented and detailed in Appendix A, together with other developed applications for demonstration purposes. Unfortunately, due to time constraints, the StreamART2A/SOM was not included in the contributed software, but will be in the near future.

### 8.1.8 Overall Assessment

A properly converged SOM model, either generated from the StreamART2A/SOM methodology or continuously made available by the UbiSOM algorithm, may be the most compact and rich abstraction of a multi-dimensional data stream of all current data stream clustering methods. From this model, visual inferences of the natural clusters and cluster descriptions can be performed; the visualizations are understandable by laymen without any prior expertise or knowledge in data mining. Additionally, as demonstrated is versatile in allowing both clustering of observations and features — consequently, also for time series.

Regarding the enumerated contributions in Section 8.1.1, contributions *ii.)* and *iii.)* fill a gap in the current state-of-the-art of data streams clustering methods by targeting self-organizing maps as an alternative solution, focused on visual exploratory cluster analysis. Consequently, regarding the previous panorama reviewed in Section 2.7, and later summarized in Figure 2.10, following this research study it can be completed as illustrated in Figure 8.1. These two contributions can also be considered relevant to the ANN field by introducing variants of well-known and established algorithms, namely the ART2-A and SOM algorithms.

Regarding these particular two contributions, each has its own strengths and weaknesses that should be balanced in the intended application.

- The StreamART2A/SOM methodology exhibiting advantages in terms of ease of

Proposed methods address the use of Self-Organizing Maps over data streams. Further proposed methodologies addressed the UbiSOM in collaborative / distributed learning settings within ubiquitous environments.

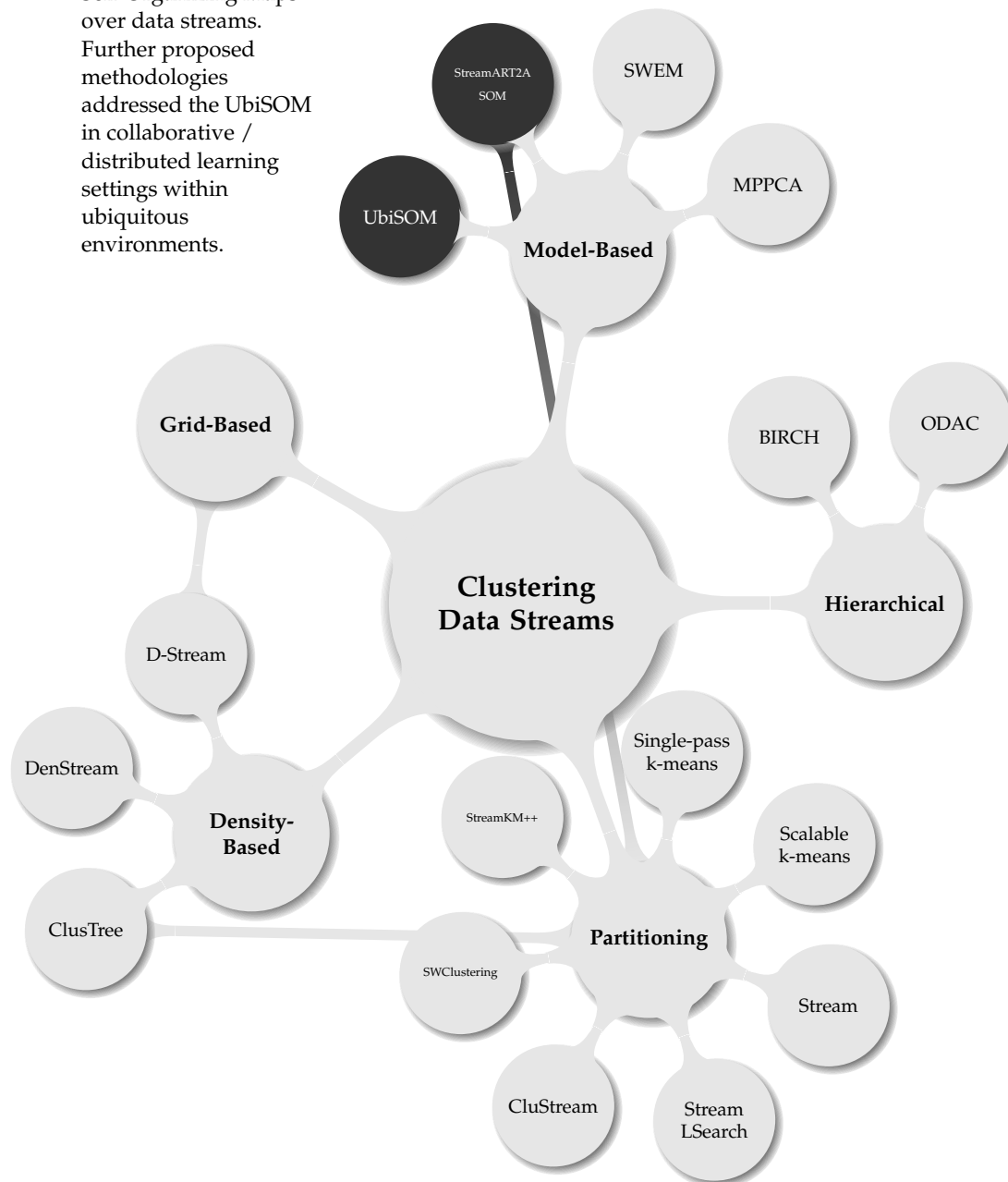


Figure 8.1: Panorama of available methods for clustering data streams after research.

parameterization and confidence in the obtained results for the specified data stream interval, since they are generated for a well-defined time interval of the data stream. On the other hand, there is inevitably a delay in selecting this time interval (that is also limited by the size of codebook) and generating the corresponding Batch SOM model. The financial real-world application of the methodology hinted on applications that can, notwithstanding, benefit from this, e.g., where knowledge discovery is intended along regular intervals of the data stream;

- The UbiSOM algorithm, in a sense, inverts the strengths and weaknesses of the later approach. The main limitations identified are the parameterization of the algorithm and confidence in the results. Regarding the former, although the parameter-space was narrowed to good overall intervals, its optimal performance relies on a more careful parameterization; to this extent, and if possible, a PSA should be performed with available prior data. Concerning the later limitation, besides the trend of the *drift* function, the proposal of additional visualizations, e.g., *BMU* and *Activity Map* visualizations should give a good visual indication of the convergence of the map regarding the recency of the information quantized. However, assuming a good parameterization the UbiSOM excels in providing a continuous SOM abstraction of the data stream from where the visualizations can be presented in real-time. The UbiSOM relies on global assessment metrics to guide its learning parameters and will describe only the current underlying distribution in the data stream. This contrasts with the StreamART2A/SOM approach, where a specific time interval may contain a mixture of different underlying distributions contained in the data stream during the specified interval.

Contribution *iv.*), involving methodologies to tackle ubiquitous data streams, still leave several open issues, but can be regarded as a solid starting point for future developments. The real-world application involving the *distributed learning* strategy seems very sound in practice. However, the *collaborative learning* strategy still has to be demonstrated in the real-world, targeting, e.g., participatory sensing data.

Finally, from the real-world applications of contribution *v.*), what stands out is the clear applicability of the proposed methods to concrete real-world application scenarios and their versatility in allowing clustering of observations and features.

## 8.2 Future Work and Open Issues

Research work is never concluded, it is always only a step forward towards more advanced research topics. Thus, this research study leaves some unsolved problems and opens new questions to which additional research effort should be devoted in the future. These are described in the following paragraphs, although at the end of each chapter devoted to the main contributions, namely in Sections 4.6, 5.8, 6.5 and 7.6, these particular improvements or further developments have already been addressed in more detail.



However, some stand out and are also recalled in this section.

**Future practical applications and collaborations.** The practical application of the contributions made in this thesis is the main goal in respect to future work. To this extent, the following are identified:

- Some existing collaborations should be reinforced in respect to the financial domain, e.g., with *GoBusiness Finance*, and others relating to sensor data should be pursued, namely with EDP and APA, following the exemplified application scenarios in Chapter 7;
- Interactive visualization of high-dimensional streaming data is a path to be further explored, as introduced in (Marques, Silva, and Santos 2016);
- In general, the IoT offers endless possibilities in which to apply the proposals and related methodologies, particularly with the advent of 5G networks and increasing connectivity between all sorts of, e.g., sensors, gadgets and smart appliances. Health care and automotive applications with ever increasing monitoring data also seem interesting;
- Finally, the UbiSOM algorithm should be studied as a viable solution for knowledge discovery from *Big Data*, i.e., as a single-pass approximation algorithm for large amounts of data. Note that in this particular application data should be considered static (therefore, stationary) by randomizing the order of the observations. Such contrasts with a single-pass of the Online SOM algorithm, in the sense that the UbiSOM algorithm provides a natural convergence to the data instead of the convergence imposed by annealing schemes over the learning parameters. This is relevant in such applications where knowledge discovery in a timely manner is of concern.

To successfully pursue the above practical application scenarios, there are still some open research issues that should be addressed, as discussed in the following paragraphs.

**Automatic change detection methods.** For example, the inclusion of an automatic change detection mechanism, either in the StreamART2A or the UbiSOM algorithm, would allow the following:

- A general alert mechanism to signal change in the data stream;
- By consequence, allow snapshots of current codebooks to be stored for later analysis and comparison with subsequent ones.

Regarding the distributed and collaborative learning strategies, it would also allow:



- In the distributed learning methodology, if the concept of the data being produced a local node is stable, then the codebooks will converge and/or stabilize, and transmissions will become redundant. Hence, only when change is detected should the node inform the central location and transmit the new filtered codebook, increasing the scalability of the solution by reducing the communication needs (Gama, Rodrigues, and Lopes 2011). This, however, requires the central location to maintain the current codebooks of each local node separately, so they can be replaced when local change occurs;
- In the *collaborative learning* strategy, at the moment, for a specific node, the local model is only incorporated in the global model when a collaboration procedure (with other node) is triggered. Automatic change detection over the local model would allow this incorporation to be made whenever change occurs, more precisely after the local UbiSOM has converged to the new distribution.

At the end of Chapter 4 some hints for the development of non-parametric methods that use the  $\overline{qe}(t)$  or the  $d(t)$  functions were discussed and can be pursued, as well as other independent methods that can be incorporated.

**Data Normalization.** Normalization of data streams requires that lower and upper bounds of individual features values are known. In sensor data, for example, data is bounded to the range of the sensor outputs themselves. In financial data, one can establish bounds from historical information or within a window model. Nonetheless, data stream normalization is critical aspect that should be addressed in future work so as to improve the practical deployment of the proposed methods.

**Noisy, Missing and Spatial Data.** Real-world data acquired on the fly, e.g., sensor and financial data, and without any preprocessing, is inherently noisy and/or incomplete. Some recommendations were made regarding dealing with noise in the StreamART2A algorithm and the UbiSOM algorithm implicitly includes some tolerance to noise. Nonetheless, an effective study of the sensitivity to noise of both algorithms should be made. Missing data is an old problem that has already been addressed for the SOM in general. Fortunately, as discussed at the end of Chapter 7, existing strategies can be easily included in the UbiSOM algorithm and extrapolated to the StreamART2A algorithm (both have similar *match* and *update* rules). For the simplest cases a preprocessing engine can perform simple *last observation carried forward* substitutions on missing feature values.

Dealing with spatial data, e.g., geographic information, included in the streaming observations requires different *match* and *update* rules for competitive learning algorithms. For example, in the SOM, the topology of the map should be maintained for the spatial data while projecting the remaining dimensions. As discussed, the inclusion of spatial data could enable the application of the proposed collaborative learning strategies to

more meaningful scenarios related to participatory sensing. To this extent, the application of current strategies discussed in Section 7.6 can be pursued.

**Heterogeneous Data.** Here, the term “heterogeneous data” refers to observations containing features with mixed data types, e.g., numerical and categorical; by extension, spatial data can also be included, but was discussed above separately. Although the StreamART2A and UbiSOM algorithms were proposed for real-valued features, there are proposals in literature, e.g., for the SOM, to additionally process categorical data. If such data is of interest for the presented methods, this path can be explored.

**Other Improvements and Evaluations.** Finally, additional interesting improvements and evaluations that can be pursued are enumerated next:

- Additional mechanisms to inform the user of the quality of the UbiSOM convergence to the underlying data stream;
- Dynamic adjustment of  $\beta$  and  $T$  in the UbiSOM algorithm to achieve an UbiSOM variant with reduced parameter-space;
- Overall improvement of the collaborative learning strategy and study of forgetting mechanisms for the methodology.

### 8.3 Final Remark

Towards the general problematic of using Self-Organizing Maps for exploratory cluster analysis over ubiquitous data streams, this thesis made original, interesting and experimentally proven contributions, contextualized within the ANN field. These extend the current state-of-the-art regarding data stream clustering methods. Also, the ubiquitous aspect of data streams was addressed leveraging the trade-off between local and global models. Regarding existing methods, the contributions favor visual data exploration and versatility in targeting different clustering problems, i.e., observations and features, with the same method. A large number of interesting research opportunities is still left open by this work, which will, hopefully, stimulate further advancements.

# Bibliography

- Achtert, E. et al. (2012). "Evaluation of Clusterings - Metrics and Visual Support". In: *IEEE 28th International Conference on Data Engineering (ICDE 2012)*, pp. 1285–1288.
- Ackermann, M. R. et al. (2012). "StreamKM++: A clustering algorithm for data streams". *Journal of Experimental Algorithmics (JEA)*, 17, pp. 2–4.
- Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R. (2005). "Geometric approximation via coresets". *Combinatorial and computational geometry*, 52, pp. 1–30.
- Agência Portuguesa do Ambiente. *QualAr website*. Retrieved on 2016/05/19 at <http://qualar.apambiente.pt>.
- Aggarwal, C. et al. (2003). "A framework for clustering evolving data streams". In: *Proceedings of the 29th International Conference on Very Large Databases*. Vol. 29. Morgan Kaufmann Publishers Inc., pp. 81–92.
- Aggarwal, C. C. (2007). *Data streams: models and algorithms*. Vol. 31. Springer Science & Business Media.
- Agrawal, R. et al. (1998). "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pp. 94–105.
- Allende, H. et al. (2004). "Robust self-organizing maps". In: *Progress in Pattern Recognition, Image Analysis and Applications*. Springer, pp. 179–186.
- Amini, A., Wah, T. Y., and Saboochi, H. (2014). "On density-based data streams clustering algorithms: A survey". *Journal of Computer Science and Technology*, 29(1), pp. 116–141.
- Antonellis, P., Makris, C., and Tsirakis, N. (2009). "Algorithms for clustering clickstream data". *Information Processing Letters*, 109(8), pp. 381–385.
- Babcock, B. et al. (2002). "Models and issues in data stream systems". In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1–16.
- Bação, F., Lobo, V., and Painho, M. (2005a). "Self-organizing Maps as Substitutes for K-Means Clustering". In: *5th International Conference on Computational Science*.
- Bação, F., Lobo, V., and Painho, M. (2005b). "The self-organizing map, the Geo-SOM, and relevant variants for geosciences". *Computers & Geosciences*, 31(2), pp. 155–163.

- Baço, F., Lobo, V., and Painho, M. (2008). "Applications of different self-organizing map variants to geographical information science problems". In: *Self-Organising Maps: applications in geographic information science*. Ed. by P. Agarwal and A. Skupin. John Wiley & Sons, Ltd. Chap. 2, pp. 21–44.
- Bandyopadhyay, S. et al. (2006). "Clustering distributed data streams in peer-to-peer environments". *Information Sciences*, 176(14), pp. 1952–1985.
- Baraldi, A. and Blonda, P. (1999). "A survey of fuzzy clustering algorithms for pattern recognition. II". *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(6), pp. 786–801.
- Barbará, D. (2002). "Requirements for clustering data streams". In: *ACM SIGKDD Explorations Newsletter*. Vol. 3. 2. ACM, pp. 23–27.
- Bauer, H.-U. and Pawelzik, K. R. (1992). "Quantifying the neighborhood preservation of self-organizing feature maps". *Neural Networks, IEEE Transactions on*, 3(4), pp. 570–579.
- Berglund, E. (2010). "Improved PLSOM algorithm". *Applied Intelligence*, 32(1), pp. 122–130.
- Berkhin, P. (2006). "A survey of clustering data mining techniques". In: *Grouping multidimensional data*. Springer, pp. 25–71.
- Blinchikoff, H. and Zverev, A. (2001). *Filtering in the Time and Frequency Domains*. Classic series. Institution of Engineering and Technology. ISBN: 9781884932175.
- Bošnjak, S. and Cisar, S. M. (2010). "EWMA Based Threshold Algorithm for Intrusion Detection". *Computing and Informatics*, 29, pp. 1089–1101.
- Bradley, P. S., Fayyad, U. M., Reina, C., et al. (1998). "Scaling Clustering Algorithms to Large Databases." In: *Knowledge Discovery in Databases*, pp. 9–15.
- Burke, J. A. et al. (2006). "Participatory sensing". *Center for Embedded Network Sensing*.
- Cao, F. et al. (2006). "Density-Based Clustering over an Evolving Data Stream with Noise." In: *SDM*. Vol. 6. SIAM, pp. 328–339.
- Carpenter, G. A. and Grossberg, S. (1988). "ART 2: Self-organization of stable category recognition codes for analog input patterns". In: *Robotics and IECON'87 Conferences*. International Society for Optics and Photonics, pp. 272–280.
- Carpenter, G. A., Grossberg, S., and Reynolds, J. H. (1991). "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network". *Neural networks*, 4(5), pp. 565–588.
- Carpenter, G. A., Grossberg, S., and Rosen, D. B. (1991a). "ART 2-A: An adaptive resonance algorithm for rapid category learning and recognition". *Neural networks*, 4(4), pp. 493–504.
- Carpenter, G. A., Grossberg, S., and Rosen, D. B. (1991b). "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system". *Neural networks*, 4(6), pp. 759–771.
- Carvalho, A et al. (2010). "High ozone levels in the northeast of Portugal: Analysis and characterization". *Atmospheric Environment*, 44(8), pp. 1020–1031.

- Chakrabarti, D., Kumar, R., and Tomkins, A. (2006). "Evolutionary clustering". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 554–560.
- Chen, N. and Marques, N. C. (2005). "An extension of self-organizing maps to categorical data". In: *12th Portuguese Conference on Artificial Intelligence*. Springer, pp. 304–313.
- Chen, Y. and Tu, L. (2007). "Density-based clustering for real-time stream data". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 133–142.
- Cormode, G., Muthukrishnan, S., and Zhuang, W. (2007). "Conquering the divide: Continuous clustering of distributed data streams". In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, pp. 1036–1045.
- Cottrell, M., Fort, J.-C., and Pagès, G. (1998). "Theoretical aspects of the SOM algorithm". *Neurocomputing*, 21(1), pp. 119–138.
- Cottrell, M. and Letrémy, P. (2005). "Missing values: Processing with the Kohonen algorithm". *Applied Stochastic Models and Data Analysis*, pp. 489–496.
- Dang, X. H. et al. (2009). "An EM-based algorithm for clustering data streams in sliding windows". In: *International Conference on Database Systems for Advanced Applications*. Springer, pp. 230–235.
- Dash, M. and Liu, H. (2000). "Feature selection for clustering". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 110–121.
- Datta, S. et al. (2006). "Distributed data mining in peer-to-peer networks". *Internet Computing, IEEE*, 10(4), pp. 18–26.
- Demartines, P. and Héroult, J. (1997). "Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets". *IEEE Transactions on Neural Networks*, 8(1), pp. 148–154.
- Deng, D. and Kasabov, N. (2003). "On-line pattern analysis by evolving self-organizing maps". *Neurocomputing*, 51, pp. 87–103.
- DeSieno, D. (1988). "Adding a conscience to competitive learning". In: *Neural Networks, 1988., IEEE International Conference on*. IEEE, pp. 117–124.
- Dutta, P. et al. (2009). "Common sense: participatory urban sensing using a network of handheld air quality monitors". In: *Proceedings of the 7th ACM conference on embedded networked sensor systems*. ACM, pp. 349–350.
- Eletricidade de Portugal (2016). *EDP RE:DY website*. Retrieved on 26/06/2016 from <https://energia.edp.pt/particulares/servicos/redy>.
- Erwin, E., Obermayer, K., and Schulten, K. (1992). "Self-organizing maps: Ordering, convergence properties and energy functions". *Biol. Cybern.* 67(1), pp. 47–55.
- Ester, M. et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining – (KDD 96)*. Vol. 96. 34, pp. 226–231.
- Farnstrom, F., Lewis, J., and Elkan, C. (2000). "Scalability for clustering algorithms revisited". In: *ACM SIGKDD Explorations Newsletter*. Vol. 2. 1. ACM, pp. 51–57.

- Fort, J. C., Letremy, P., and Cottrel, M. (2002). "Advantages and drawbacks of the Batch Kohonen algorithm". In: *10th-European-Symposium on Artificial Neural Networks. Esann'2002. Proceedings. 2002*: 223-30.
- Frahling, G. and Sohler, C. (2008). "A fast k-means implementation using coresets". *International Journal of Computational Geometry & Applications*, 18(06), pp. 605-625.
- Fritzke, B. (1994). "Growing cell structures — a self-organizing network for unsupervised and supervised learning". *Neural networks*, 7(9), pp. 1441-1460.
- Fritzke, B. (1997). *Some competitive learning methods*. Tech. rep. Artificial Intelligence Institute, Dresden University of Technology.
- Fritzke, B. et al. (1995). "A growing neural gas network learns topologies". *Advances in neural information processing systems*, 7, pp. 625-632.
- Furao, S. and Hasegawa, O. (2006). "An incremental network for on-line unsupervised classification and topology learning". *Neural Networks*, 19(1), pp. 90-106.
- Furao, S., Ogura, T., and Hasegawa, O. (2007). "An enhanced self-organizing incremental neural network for online unsupervised learning". *Neural Networks*, 20(8), pp. 893-903.
- Gaber, M. M. et al. (2014). "Data stream mining in ubiquitous environments: state-of-the-art and current directions". *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(2), pp. 116-138.
- Gama, J. (2010). *Knowledge discovery from data streams*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series.
- Gama, J., Rodrigues, P. P., and Lopes, L. (2011). "Clustering distributed sensor data streams using local processing and reduced communication". *Intelligent Data Analysis*, 15(1), pp. 3-28.
- Gama, J. et al. (2004). "Learning with drift detection". In: *Advances in Artificial Intelligence—SBIA 2004*. Springer, pp. 286-295.
- García, H. L. and González, I. M. (2004). "Self-organizing map and clustering for wastewater treatment monitoring". *Engineering Applications of Artificial Intelligence*, 17(3), pp. 215-225.
- GmbH, V. S. *Viscovery SOMine 6.0*. URL: <http://www.eudaptics.com/>.
- GoBusiness Finance (2016). *GoBusiness Finance website*. Retrieved on 20/04/2016 from <https://gobusinessfinance.ch/en/>.
- Grossberg, S. (1976). "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors". *Biological cybernetics*, 23(3), pp. 121-134.
- Guha, S. et al. (2003). "Clustering data streams: Theory and practice". *IEEE Transactions on Knowledge and Data Engineering*, pp. 515-528.
- Guha, S. et al. (2000). "Clustering data streams". In: *41st annual symposium on Foundations of computer science*. IEEE, pp. 359-366.
- Han, J., Kamber, M., and Pei, J. (2006). *Data mining: concepts and techniques*. Morgan kaufmann.

- Hastie, T. and Stuetzle, W. (1989). "Principal curves". *Journal of the American Statistical Association*, 84(406), pp. 502–516.
- Ho, S.-S. and Wechsler, H. (2007). "Detecting Changes in Unlabeled Data Streams Using Martingale." In: *International Joint Conference on Artificial Intelligence*, pp. 1912–1917.
- Ilango, M. and Mohan, V (2010). "A survey of grid based clustering algorithms". *International Journal of Engineering Science and Technology*, 2(8), pp. 3441–3446.
- James, G. et al. (2013). *An introduction to statistical learning*. Vol. 6. Springer.
- Kargupta, H. et al. (2004). "Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring". In: *Proceedings of SIAM International Conference on Data Mining*. Vol. 334.
- Kargupta, H. et al. (2001). "Distributed clustering using collective principal component analysis". *Knowledge and Information Systems*, 3(4), pp. 422–448.
- Kargupta, H. et al. (2002). "MobiMine: Monitoring the stock market from a PDA". *ACM SIGKDD Explorations Newsletter*, 3(2), pp. 37–46.
- Kaski, S. (1997). "Data exploration using self-organizing maps". PhD thesis. Helsinki University of Technology.
- Kaufman, L. and Rousseeuw, P. (1987). "Clustering by means of medoids". *Statistical Data Analysis*.
- Keim, D. A. (2002). "Information visualization and visual data mining". *IEEE Transactions on Visualization and Computer Graphics*, 8(1), pp. 1–8.
- Keogh, E. and Lin, J. (2005). "Clustering of time-series subsequences is meaningless: implications for previous and future research". *Knowledge and information systems*, 8(2), pp. 154–177.
- Kifer, D., Ben-David, S., and Gehrke, J. (2004). "Detecting change in data streams". In: *Proceedings of the Thirtieth international conference on Very large data bases*. VLDB Endowment, pp. 180–191.
- Kirk, J. S. and Zurada, J. M. (1999). "Algorithms for improved topology preservation in self-organizing maps". In: *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 3. Piscataway, NJ: IEEE Service Center, pp. 396–400.
- Klusch, M., Lodi, S., and Moro, G. (2003). "Distributed clustering based on sampling local density estimates". In: *IJCAI*, pp. 485–490.
- Kohonen, T. (1982). "Self-organized formation of topologically correct feature maps". *Biological cybernetics*, 43(1), pp. 59–69.
- Kohonen, T. (1996). *SOM\_PAK - The Self Organizing Map Program Package*. Tech. rep. Helsinki University of Technology, Laboratory of Computer and Information Science.
- Kohonen, T. (2001). *Self-organizing maps*. Vol. 30. Springer Science & Business Media.
- Kohonen, T. (2013). "Essentials of the self-organizing map". *Neural Networks*, 37, pp. 52–65.



- Kontaki, M., Papadopoulos, A. N., and Manolopoulos, Y. (2008). "Continuous trend-based clustering in data streams". In: *International Conference on Data Warehousing and Knowledge Discovery*. Springer, pp. 251–262.
- Kranen, P. et al. (2011). "The ClusTree: indexing micro-clusters for anytime stream mining". *Knowledge and information systems*, 29(2), pp. 249–272.
- Kruskal, J. (1964). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". *Psychometrika*, 29(1), pp. 1–27.
- Lee, S.-C., Gu, J.-C., and Suh, Y.-H. (2006). "A comparative analysis of clustering methodology and application for market segmentation: K-means, SOM and a two-level SOM". In: *Foundations of Intelligent Systems*. Springer, pp. 435–444.
- Li, Q. et al. (2011). "Continuously identifying representatives out of massive streams". In: *International Conference on Advanced Data Mining and Applications*. Springer, pp. 229–242.
- Liao, T. W. (2005). "Clustering of time series data — a survey". *Pattern recognition*, 38(11), pp. 1857–1874.
- Lichman, M. (2013). *UCI Machine Learning Repository*. URL: <http://archive.ics.uci.edu/ml>.
- Linde, Y., Buzo, A., and Gray, R. (1980). "An algorithm for vector quantizer design". *Communications, IEEE Transactions on*, 28(1), pp. 84–95.
- Liu, B., Xia, Y., and Yu, P. S. (2000). "Clustering through decision tree construction". In: *Proceedings of the ninth international conference on Information and knowledge management*. ACM, pp. 20–29.
- MacQueen, J. et al. (1967). "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA., pp. 281–297.
- Malsburg, C. Von der (1973). "Self-organization of orientation sensitive cells in the striate cortex". *Kybernetik*, 14(2), pp. 85–100.
- Marler, R. T. and Arora, J. S. (2004). "Survey of multi-objective optimization methods for engineering". *Structural and multidisciplinary optimization*, 26(6), pp. 369–395.
- Marques, N. C., Silva, B., and Santos, H. (2016). "An Interactive Interface for Multidimensional Data Stream Analysis". In: *2016 20th International Conference Information Visualisation (IV)*, pp. 223–229. DOI: [10.1109/IV.2016.72](https://doi.org/10.1109/IV.2016.72).
- Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). "Neural-gas' network for vector quantization and its application to time-series prediction". *Neural Networks, IEEE Transactions on*, 4(4), pp. 558–569.
- Matos, D., Marques, N. C., and Cardoso, M. (2014). "Agrupamento sobre uma matriz de distâncias UMAT – uma aplicação sobre dados financeiros". In: *Proceedings of the XXI Jornadas de Classificação e Análise de Dados*.
- McLachlan, G. and Krishnan, T. (2007). *The EM algorithm and extensions*. Vol. 382. John Wiley & Sons.



- Melnykov, V., Maitra, R., et al. (2010). "Finite mixture models and model-based clustering". *Statistics Surveys*, 4, pp. 80–116.
- Neme, A. and Hernández, L. (2011). "Visualizing patterns in the air quality in mexico city with self-organizing maps". In: *Advances in Self-Organizing Maps*. Springer, pp. 318–327.
- O'callaghan, L. et al. (2002). "Streaming-data algorithms for high-quality clustering". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, pp. 0685–0685.
- Ogasawara, E. et al. (2010). "Adaptive normalization: A novel data normalization approach for non-stationary time series". In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, pp. 1–8.
- Organization, W. H. (1999). *Monitoring ambient air quality for health impact assessment*.
- Pan, S. J. and Yang, Q. (2010). "A survey on transfer learning". *Knowledge and Data Engineering, IEEE Transactions on*, 22(10), pp. 1345–1359.
- Pearce, J. L. et al. (2014). "Using self-organizing maps to develop ambient air quality classifications: a time series example". *Environmental Health*, 13(1), p. 56.
- Penn, B. S. (2005). "Using self-organizing maps to visualize high-dimensional data". *Computers & Geosciences*, 31(5), pp. 531–544. ISSN: 0098-3004.
- Pérez-Uribe, A. et al. (2007). "Improving the correlation hunting in a large quantity of SOM component planes". In: *Artificial Neural Networks – ICANN 2007*. Springer, pp. 379–388.
- Pöllä, M., Honkela, T., and Kohonen, T. (2009). "Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum". *Neural Computing Surveys*.
- Polzlbauer, G. (2004). "Survey and Comparison of Quality Measures for Self-Organizing Maps". In: *Fifth Workshop on Data Analysis – WDA 2004*. Elfa Academic Press, Kosice, pp. 67–82.
- Rauber, A., Merkl, D., and Dittenbach, M. (2002). "The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data". *IEEE Transactions on Neural Networks*, 13(6), pp. 1331–1341.
- Rodrigues, P. P. and Gama, J. (2014). "Distributed clustering of ubiquitous data streams". *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(1), pp. 38–54.
- Rodrigues, P. P., Gama, J., and Lopes, L. (2008). "Clustering distributed sensor data streams". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 282–297.
- Rodrigues, P. P., Gama, J., and Pedroso, J. P. (2008). "Hierarchical clustering of time-series data streams". *Knowledge and Data Engineering, IEEE Transactions on*, 20(5), pp. 615–627.
- Rosenblatt, F. (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Spartan Books.
- Rougier, N. and Boniface, Y. (2011). "Dynamic self-organising map". *Neurocomputing*, 74(11), pp. 1840–1847.

- Rumelhart, D. E. and Zipser, D. (1985). "Feature discovery by competitive learning". *Cognitive science*, 9(1), pp. 75–112.
- Rustum, R. and Adeloye, A. J. (2007). "Replacing outliers and missing values from activated sludge data using Kohonen self-organizing map". *Journal of Environmental Engineering*, 133(9), pp. 909–916.
- Samad, T. and Harp, S. A. (1992). "Self-organization with partial data". *Network: Computation in Neural Systems*, 3(2), pp. 205–212.
- Samarasinghe, S. (2006). *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. CRC Press.
- Sammon J.W., J. (1969). "A Nonlinear Mapping for Data Structure Analysis". *Transactions on Computers*, C-18(5), pp. 401–409.
- Schatten, R., Goerke, N., and Eckmiller, R. (2005). "Regional and online learnable fields". In: *Pattern Recognition and Image Analysis*. Springer, pp. 74–83.
- Shalabi, L. A., Shaaban, Z., and Kasasbeh, B. (2006). "Data mining: A preprocessing engine". *Journal of Computer Science*, 2(9), pp. 735–739.
- Silva, B. and Marques, N. C. (2007). "A hybrid parallel SOM algorithm for large maps in data-mining". In: *New Trends in Artificial Intelligence*.
- Silva, B. and Marques, N. C. (2010a). "Feature Clustering with Self-organizing Maps and an Application to Financial Time-series for Portfolio Selection". In: *Proceedings of International Conference of Neural Computation*, pp. 301–309.
- Silva, B. and Marques, N. C. (2010b). "Ubiquitous data-mining with self-organizing maps". In: *Workshop on Ubiquitous Data Mining in conjunction with the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pp. 45–50. URL: <http://www.inescporto.pt/~jgama/udm2010/procUDMECAI2010.pdf#page=51>.
- Silva, B. and Marques, N. C. (2012). "Neural Network-based Framework for Data Stream Mining". In: *Proceedings of the Sixth Starting AI Researchers' Symposium (ECAI 2012)*. Vol. 241. Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 294–305. URL: <http://www.iospress.nl/book/stairs-2012/>.
- Silva, B. and Marques, N. C. (2013). "Ubiquitous Self-Organizing Maps". In: *Proceedings of the 3rd Workshop on Ubiquitous Data Mining co-located with the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. ISSN: 1613-0073. CEUR Workshop Proceedings (<http://ceur-ws.org>), pp. 54–55. URL: <http://ceur-ws.org/Vol-1088/>.
- Silva, B. and Marques, N. C. (2015a). "Ubiquitous Self-Organizing Map: Learning Concept-Drifting Data Streams". In: *New Contributions in Information Systems and Technologies: Volume 1*. Ed. by A. Rocha et al. Cham: Springer International Publishing, pp. 713–722. ISBN: 978-3-319-16486-1. DOI: [10.1007/978-3-319-16486-1\\_70](https://doi.org/10.1007/978-3-319-16486-1_70).
- Silva, B., Marques, N. C., and Panosso, G. (2012). "Applying neural networks for concept drift detection in financial markets". In: *Workshop on Ubiquitous Data Mining in conjunction with the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pp. 43–47.

- Silva, B. and Marques, N. C. (2015b). "The ubiquitous self-organizing map for non-stationary data streams". *Journal of Big Data*, 2(1), pp. 1–22. ISSN: 2196-1115. DOI: [10 . 1186 / s40537-015-0033-0](https://doi.org/10.1186/s40537-015-0033-0).
- Silva, J. A. et al. (2013). "Data stream clustering: A survey". *ACM Computing Surveys (CSUR)*, 46(1), p. 13.
- Sorjamaa, A. et al. (2009). "Sparse linear combination of SOMs for data imputation: Application to financial database". In: *Advances in Self-Organizing Maps*. Springer, pp. 290–297.
- Stahl, F. et al. (2010). "Pocket data mining: towards collaborative data mining in mobile computing environments". In: *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*. Vol. 2. IEEE, pp. 323–330.
- Strehl, A. and Ghosh, J. (2003). "Cluster ensembles—a knowledge reuse framework for combining multiple partitions". *The Journal of Machine Learning Research*, 3, pp. 583–617.
- Tiple, P., Cavique, L., and Marques, N. C. (2016). "Ramex Forum: a tool for displaying and analysing complex sequential patterns of financial products". *Expert Systems*.
- Utsch, A. (1995). "Self Organizing Neural Networks perform different from statistical k-means clustering". In: *In Proceedings of GfKI '95*.
- Utsch, A. and Siemon, H. (1990). "Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis". In: *Proceedings of the International Neural Network Conference (INNC-90)*. Kluwer Academic Press, pp. 305–308.
- Vatsavai, R. R. et al. (2010). *Knowledge Discovery from Sensor Data*. Springer.
- Venna, J. and Kaski, S. (2001). "Neighborhood preservation in nonlinear projection methods: An experimental study". In: *Artificial Neural Networks – ICANN 2001*. Springer, pp. 485–491.
- Vesanto, J. and Ahola, J. (1999). "Hunting for Correlations in Data Using the Self-Organizing Map". In: *Proceeding of the International ICSC Congress on Computational Intelligence Methods and Applications (CIMA '99)*. Ed. by H. Bothe et al. ICSC Academic Press, pp. 279–285.
- Vesanto, J. and Alhoniemi, E. (2000). "Clustering of the self-organizing map". *Neural Networks, IEEE Transactions on*, 11(3), pp. 586–600.
- Vesanto, J., Sulkava, M., and Hollmen, J. (2003). "On the Decomposition of the Self-Organizing Map Distortion Measure". In: *Proceedings on the Workshop on Self-Organizing Maps (WSOM'03)*, pp. 11–16.
- Vesanto, J. et al. (1999). "Self-organizing map in Matlab: the SOM Toolbox". In: *Proceedings of the Matlab DSP conference*. Vol. 99, pp. 16–17.
- "Visual Data Mining" (2005). *Journal of Universal Computer Science*, 11(11), pp. 1749–1751.
- Ward Jr, J. H. (1963). "Hierarchical grouping to optimize an objective function". *Journal of the American statistical association*, 58(301), pp. 236–244.
- Wells, W. M. (1986). "Efficient synthesis of Gaussian filters by cascaded uniform filters". *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2), pp. 234–239.

- Wesolkowski, S. (2002). "Clustering with a mixture of self-organizing maps". In: *Proceedings of the 2002 International Joint Conference on Neural Networks*. Vol. 3. IEEE, pp. 2363–2368.
- White, D. (2013). *Visualization: Set Your Analytics Users Free*. Tech. rep. Aberdeen Group.
- Wurst, M. and Morik, K. (2007). "Distributed feature extraction in a P2P setting—a case study". *Future Generation Computer Systems*, 23(1), pp. 69–75.
- Yair, E., Zeger, K., and Gersho, A. (1992). "Competitive learning and soft competition for vector quantizer design". *IEEE Transactions on Signal Processing*, 40(2), pp. 294–309.
- Yin, J.-B. and Shen, H.-B. (2011). "Robust ART-2 neural network learning framework". In: *Proceedings of 2011 International Conference on Modelling, Identification and Control*.
- Zhang, T., Ramakrishnan, R., and Livny, M. (1996). "BIRCH: an efficient data clustering method for very large databases". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. Vol. 25. 2. ACM, pp. 103–114.
- Zhang, X. and Wang, W. (2010). "Self-adaptive change detection in streaming data with non-stationary distribution". In: *International Conference on Advanced Data Mining and Applications*. Springer, pp. 334–345.
- Zhou, A. et al. (2008). "Tracking clusters in evolving data streams over sliding windows". *Knowledge and Information Systems*, 15(2), pp. 181–214.



# Contributed Framework and Software

In the course of this research programming was a central task, either by implementing the algorithms, coding the simulations, automating the parameter sensitivity analysis and producing plots and visualizations to illustrate the data and obtained models in this manuscript. The majority of these tasks were performed with the *Java* and *R* languages and intermediate formats such as CSV.

However, this appendix presents developed software that can be readily used in the context of the previous illustrative real-world applications through the presentation of a developed framework (Section A.1) and a compact library (Section A.2) that can be used to build additional applications of the UbiSOM algorithm.

A website was created to promote collaborations with the developed software:

<http://ubisom.brunomnsilva.com/>

## A.1 Framework

The framework is depicted in Figure A.1 and was idealized with an ubiquity concept in mind, i.e., distributed instances of the UbiSOM that can communicate with each other and be queried by users. In a higher-level of abstraction it is composed by a data stream source, an instance of the UbiSOM algorithm, an optional *webserver* acting as *middleware* to the application layer. The framework was implemented in the *Java* language and details regarding each layer and implementation follows:

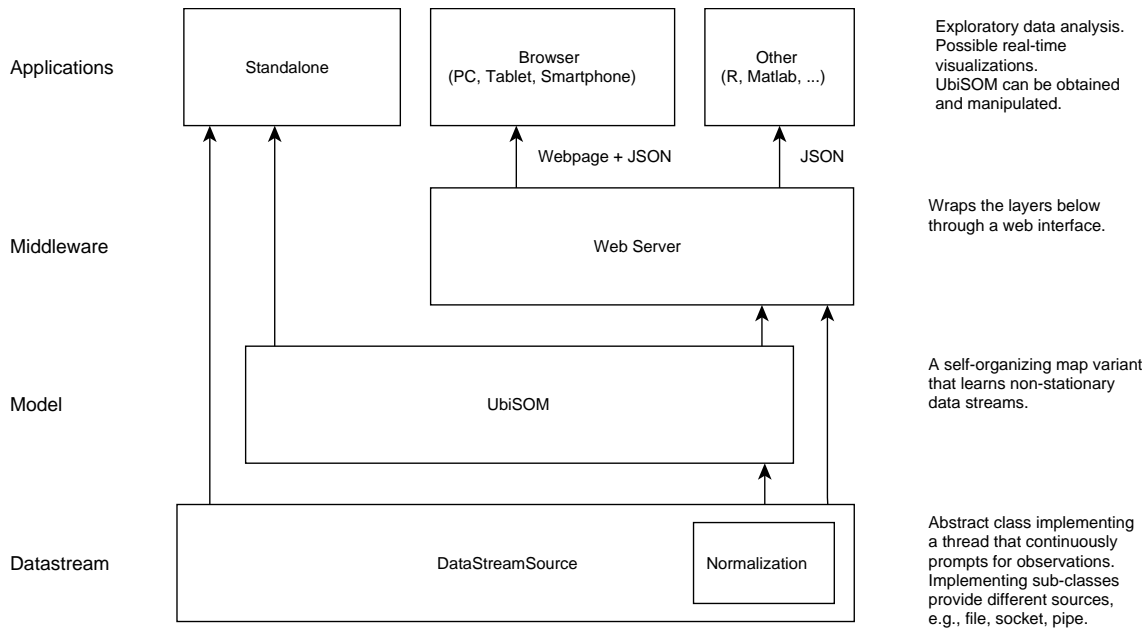


Figure A.1: Framework as a layered architecture.

**Datastream Layer.** This layer is responsible for acquiring the data stream observations and normalized them if necessary. Current implementation is based on a implemented abstract class *DataStreamSource* providing base functionality for querying for new observations and on-the-fly normalization/denormalization of patterns (observations and prototypes), given the established lower and upper limits for the features. This implementation runs in its own thread and notifies the model layer via an *Observer* software pattern.

Implementing *sub-classes* dictate how observations are acquired, e.g., from file, socket, JDBC (databases), invoking scripts that parse operating system commands, low-level communication to obtain measurements of attached sensors, etc.;

**Model Layer.** This layer is responsible for receiving the streaming observations from the layer below and producing an UbiSOM model, maintaining a topologically ordered codebook over the (possibly) non-stationary underlying data stream;

**Middleware Layer.** This layer is responsible for any module that bridges the two previous layers with the application layer. Current implementation is in the form of an *web-server*, allowing remote querying of the model. The webserver responds to requests either by sending the model in JavaScript Object Notation (JSON) embedded in a webpage that can be readily viewed in any recent browser or as raw JSON to be consumed by other applications. A specific desired time for the model can be sent, useful for synchronization if centralizing distributed models;

```

1 #####
2 ## CONFIGURATION FILE
3 #####
4 #Only way to include spaces in this properties file is to use \u0020 char
5 device.name = Household\u0020Electric\u0020Power\u0020Sensing
6
7 #####
8 # UbiSOM Parameterization
9
10 ubisom.width = 20
11 ubisom.height = 40
12 ubisom.eta_i=0.1
13 ubisom.eta_f=0.08
14 ubisom.sigma_i=0.6
15 ubisom.sigma_f=0.2
16 ubisom.beta = 0.8
17 ubisom.T = 2000
18
19 #####
20 # datastream configuration
21
22 datastream.feed.class=somlp.model.io.DataStreamSourceCSV
23 datastream.feed.args=household_power_sensor.csv
24
25 #interval in milliseconds
26 datastream.interval=50
27
28 #####
29 # webserver configuration
30 webserver.port=8000

```

Figure A.2: Configuration file for the developed framework.

**Applications Layer.** A wide variety of software applications can be created in this layer. Applications can either be built on top of the *datastream* and *model* layers or by communicating with the middleware layer. Current implementations targeted both cases.

The framework explicitly delegates different responsibilities to each layer as abstractions to the layer immediately above. The framework individual components are parameterized through a configuration file that follows the *java properties* file format. An example is given in Figure A.2 for the application regarding the Household application (Section 7.3) where data is obtained from a comma separated values (CSV) file.

The following sections briefly describe applications that were developed within this framework.

### A.1.1 Local applications

A local application is considered as software developed in a monolithic form (standalone), together with code from the *datastream* and *model* layers. A simple application that allows the real-time visualizations of the model was developed (see Figure A.3).

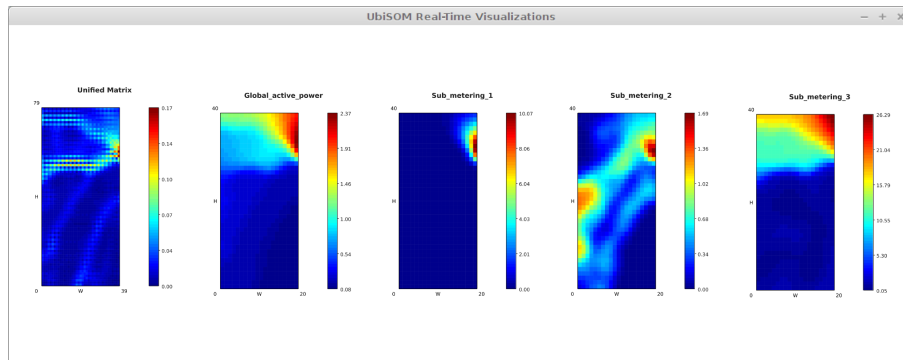


Figure A.3: Local application that allows real-time visualization of the UbiSOM model.

## A.1.2 Remote applications

These applications involve obtaining an UbiSOM model remotely and then perform exploratory data analysis or manipulate the model for further processing.

### A.1.2.1 Remote Exploratory Analysis

This application regards browser client-side model visualization and feature clustering. The client connects to the webserver at the specified port through the standard *http* protocol. The main *webpage* is returned with the current codebook embedded in JSON format; the codebook is stored locally in the browser's web storage (all recent browsers support this). Generation of the visualization and additional procedures are performed locally in the browser through *JavaScript*, freeing the server of any further computations.

In the main page the user is presented with the standard visualizations of the acquired model, i.e., U-Matrix and component planes (see Figure A.4a). From here he can opt to visualize the U-Matrix in 3D (see Figure A.4b), which allows him to detect less prominent cluster structures and to perform feature clustering (see Figure A.4c). These later functionalities are performed over the model sent with the initial webpage. It must be refreshed to obtain a more recent model.

Within this architecture a simple Android application was developed, using the built-in browser to display the visualizations (see Figure A.5).

### A.1.2.2 R Studio Integration

By using the JSON format, the models can be integrated in other applications. For example, a small library was developed in the *R* language that allows a remote model to be downloaded, visualized and manipulated. In Figure A.6 a snapshot of a Household model was obtained from the webserver and visualized in *R Studio* through this library. Interesting in the future is the integration with the *R Kohonen package* that produces additional visualizations and procedures over self-organizing map models, e.g., automatic detection of clusters via Ward's method.



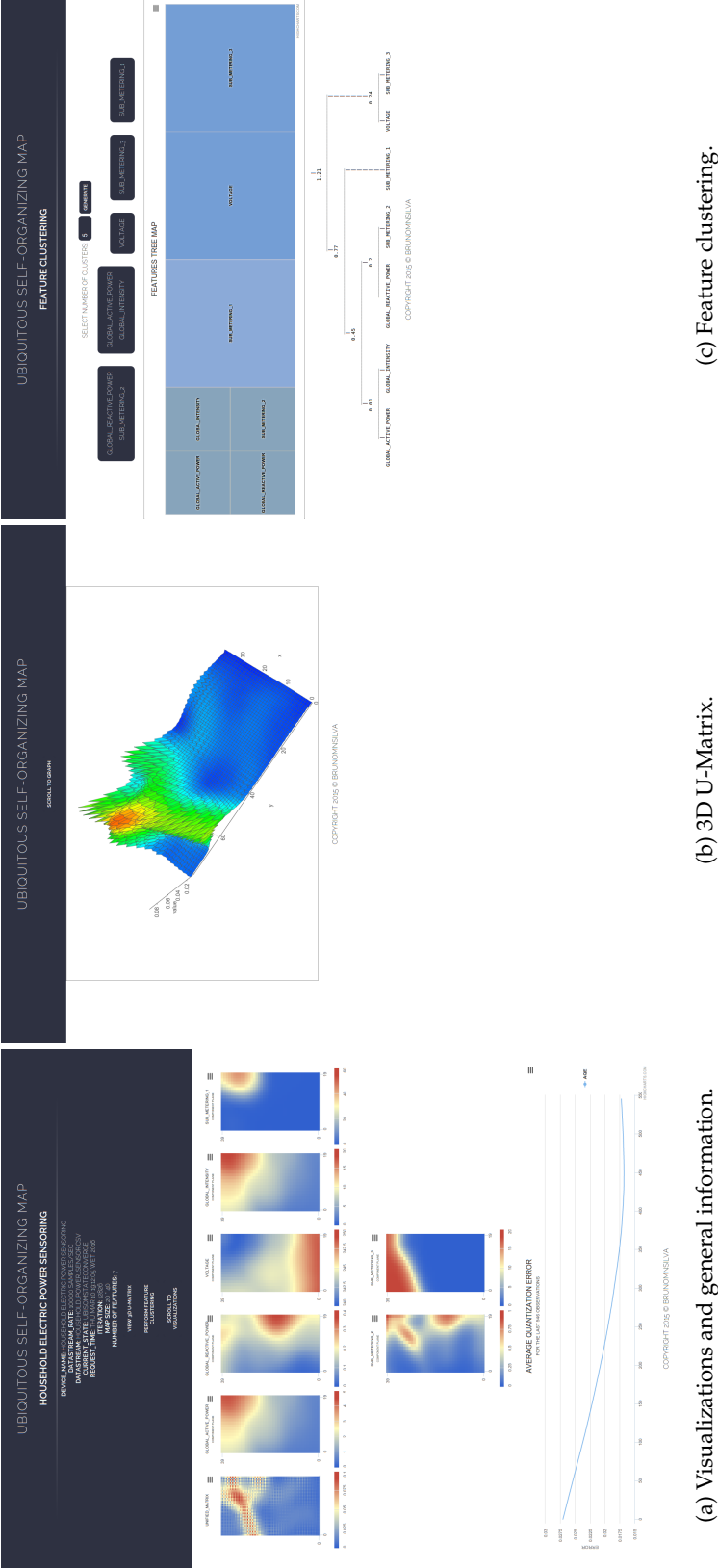


Figure A.4: Functionalities of the remote exploratory cluster analysis application.

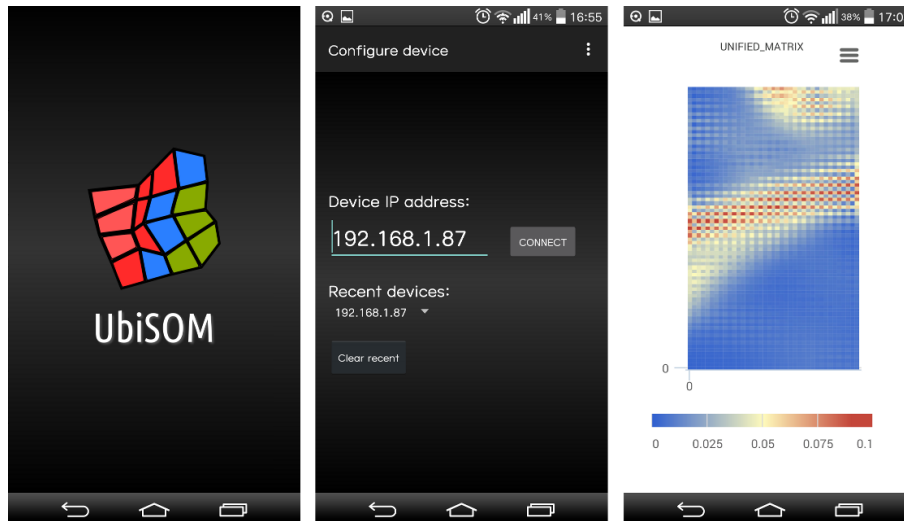


Figure A.5: Remote visualization through Android application.

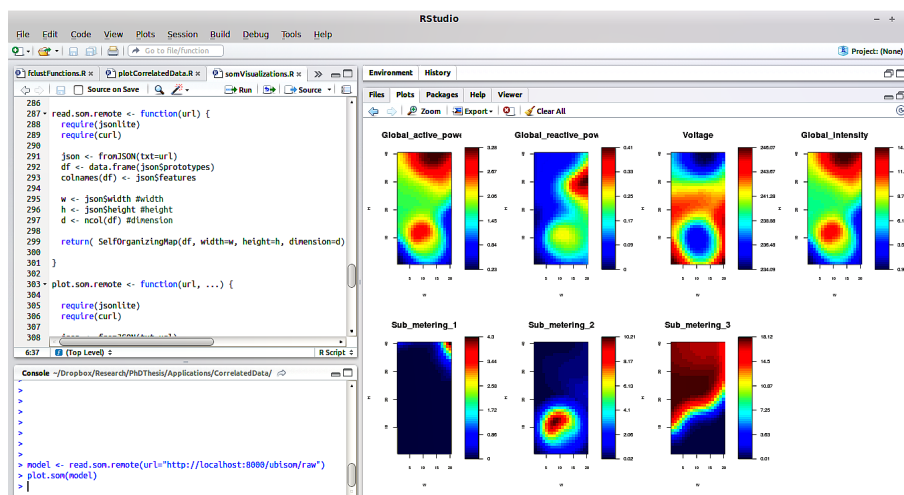


Figure A.6: R Studio integration through developed R library.

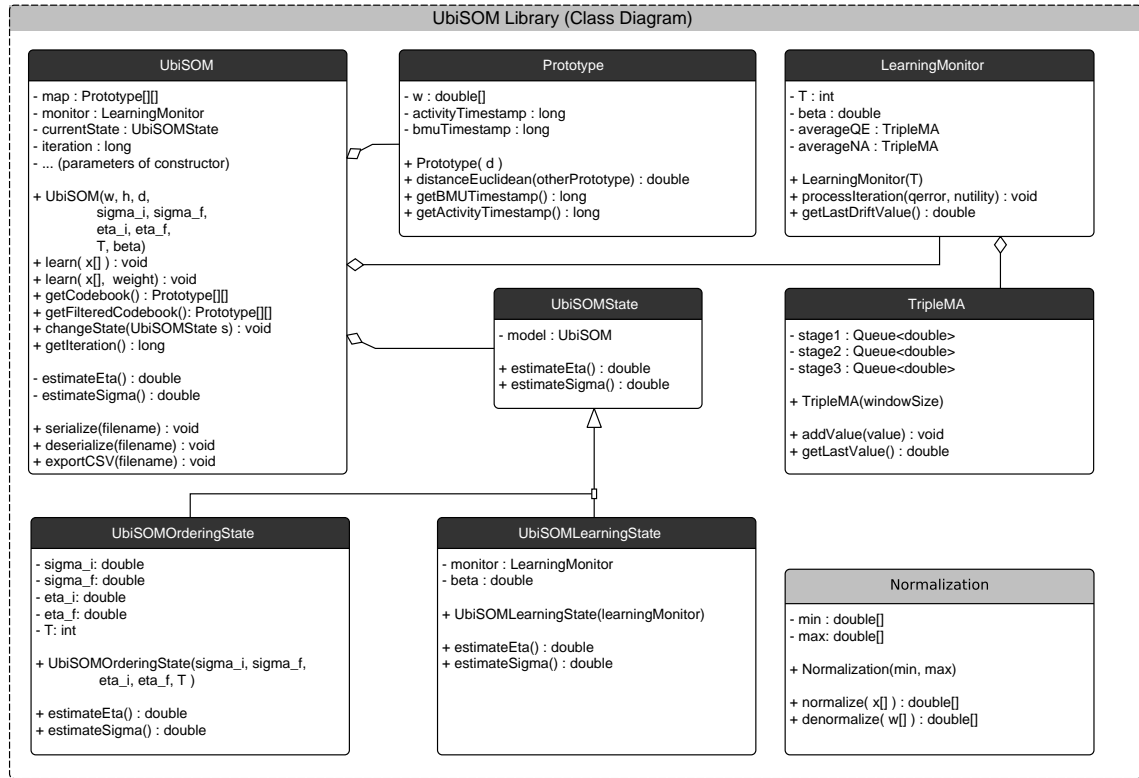


Figure A.7: UbiSOM library class diagram.

The developed R library also allows importing UbiSOM models in CSV format. It was by this method that results and graphics relating to feature clustering (Sections 5.6 and 7.5) and the component plane to air quality index mappings (Section 7.4) were obtained in this thesis.

## A.2 Library

The UbiSOM algorithm and related classes were wrapped into a standalone library that can be used to build other applications. In this context, Figure A.7 depicts the overall implementation that was effectively used throughout this research. This implementation allows the realization of the algorithm detailed in Section 5.3.

While two simple demonstrations of the usage of this UbiSOM library are presented next, it has been used to develop real-time visualization of multi-dimensional financial data streams in (Marques, Silva, and Santos 2016).

### A.2.1 Interactive Demonstration

The *UbiSOM Interactive Demo* application (see Figure A.8) allows the user to manipulate a set of Gaussian clusters in 2D space in real-time. Clusters can be created, destroyed, moved (change in mean) and their size altered (change in variance). Is is interesting

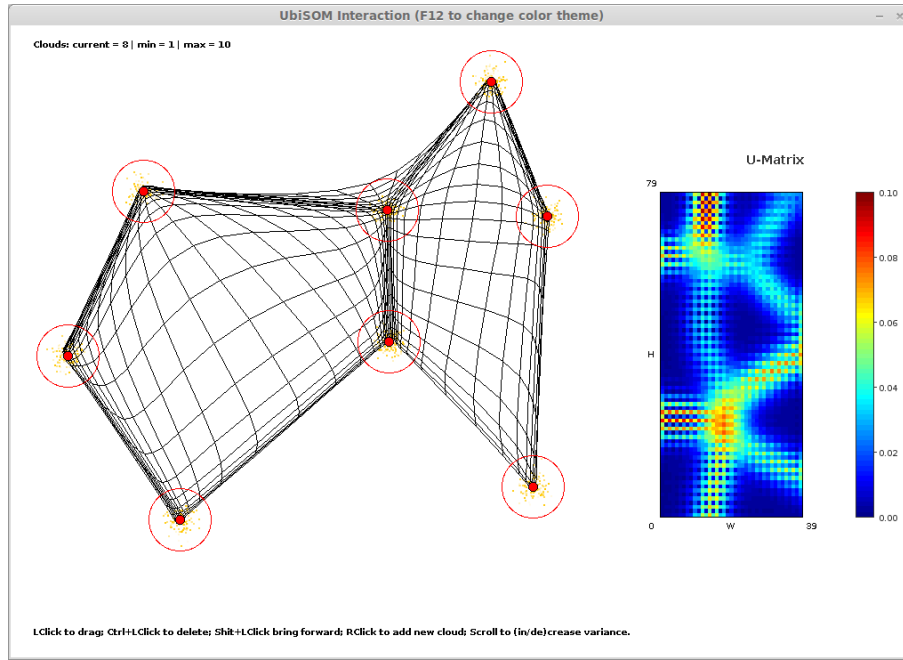


Figure A.8: UbiSOM Interactive Demo application.

for the user to observe the dynamics of the algorithm over a changing cluster structure using different parameterizations. On the right side of the application window the real-time derived U-Matrix is available, which allows to compare the created cluster structure with the one identified in the visualization.

### A.2.2 Color Quantization in a Video Stream

Finally, a demonstration of the UbiSOM ability to quantize the colors during a video stream was implemented. It illustrates the learning of a color codebook over a video stream, while creating a version of the same video but with the colors quantized.

An example using a trailer for the movie SINTEL<sup>1</sup>, produced by the *Blender Open Movie Project*, is illustrated. The video resolution is  $640 \times 360$  pixels; consequently, each frame contains 230 400 RGB color patterns ( $d = 3$ ). In each frame 2000 randomly selected pixels are normalized in the unit hypercube (RGB components vary between 0 and 255) and are presented to a  $10 \times 20$  UbiSOM instance that continuously maintains a 200-color codebook over the incoming color observations. After being presented with the color observations of the current frame, the codebook is used to produce a quantized version of the same video frame. This procedure endures until the end of the video stream. The algorithm was parameterized with  $T = 2000$  and  $\beta = 0.8$ ; other parameters remained as in Section 5.5.

Figure A.9 depicts the evolution of the UbiSOM learning, where the scene changes are clearly detected by the assessment metrics. Those periods where the learning parameters

<sup>1</sup><https://durian.blender.org/>

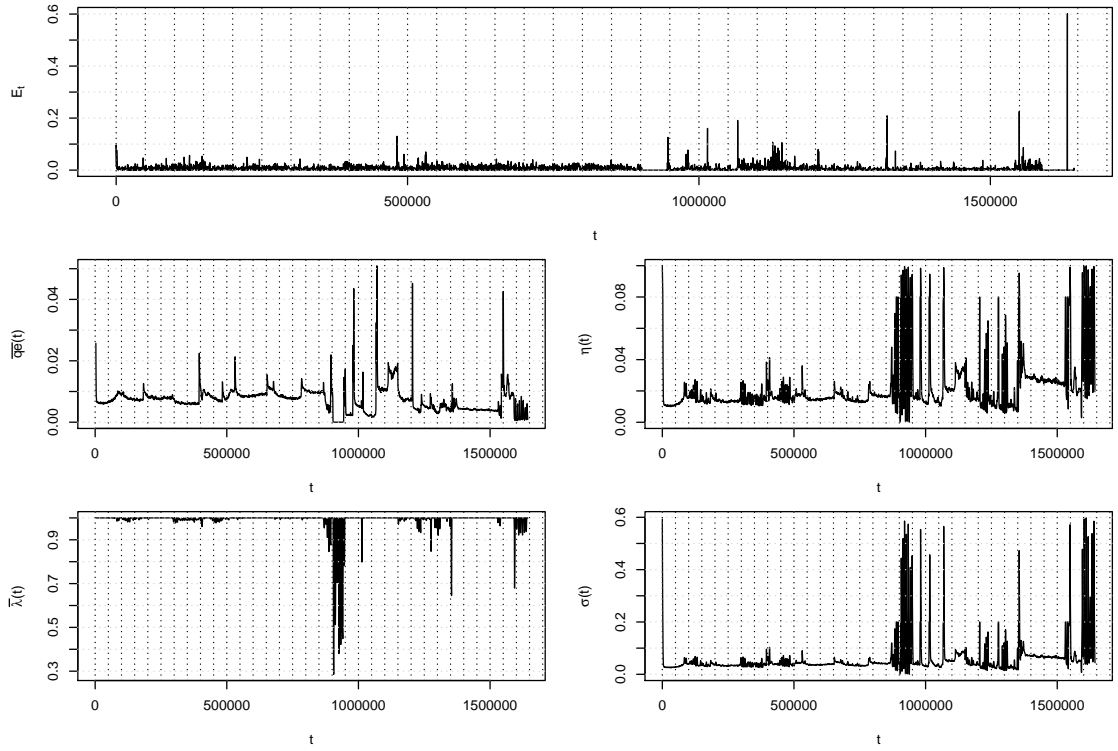
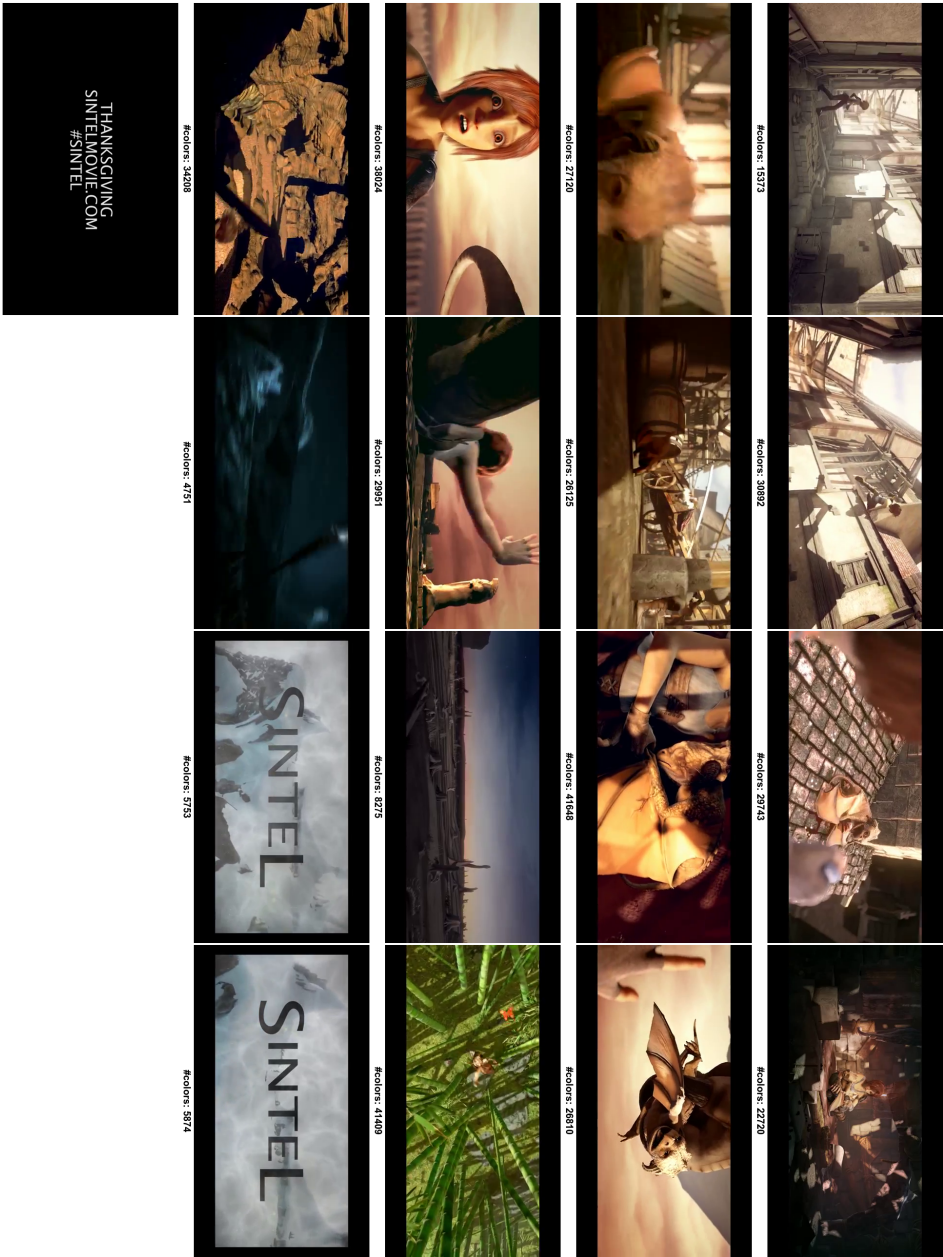


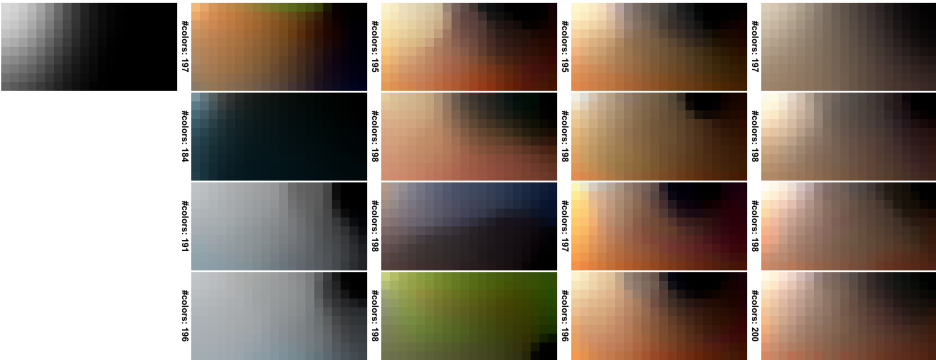
Figure A.9: Evolution of UbiSOM learning parameters over the SINTEL video frames.

seem not to converge are seconds of mostly black video frames. Figure A.10a depicts sample frames from the original video and Figure A.10b the current UbiSOM codebook after each respective frame. Both are annotated with the total number of distinct colors in the frame/codebook. The same frames from the resulting quantized video are presented in Figure A.11, also showing the total number of colors used in each frame. Please note that here the number of distinct colors exceeds in one, because of the pure white text overlay placed over the frames.



(a) SINTEL sample original video frames.

Figure A.10: SINTEL video frames and UbiSOM codebooks.



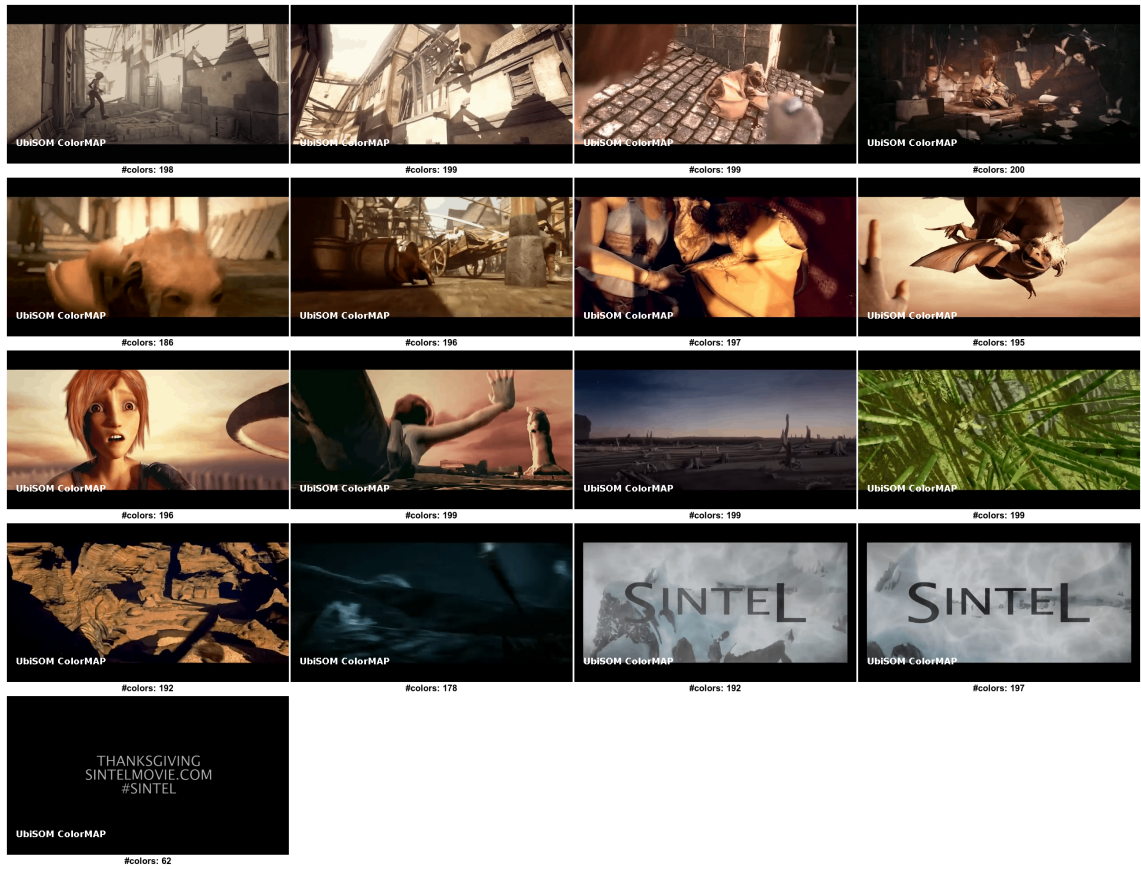


Figure A.11: Quantized SINTEL video frames by the UbiSOM algorithm.







# Artificial Data Streams

Illustrations of the artificial data streams used throughout this thesis are presented here. Table B.1 summarizes the data streams, which are described in the following sections. Besides specific aims for each data stream, all (except *Correlated*) were used in the parameter sensitivity analysis of algorithms in Chapters 4 and 5. Note that data streams (except *Correlated*) are depicted after *min-max* normalization, hence  $\mathbf{x} \in [0, 1]^d$ , i.e., all observations are within the unit hypercube.

If not explicitly stated otherwise, artificial data streams were generated by this research and are publicly available at:

<https://brunomnsilva.github.io/datastreams/>

## B.1 Gauss

Two-dimensional stationary data stream containing 100 000 points from a bi-variate normal (Gaussian) distribution, centered at the origin and identity co-variance matrix; used to evaluate if proposed and compared methods can model the input space density properly — illustrated in Figure B.1a.

## B.2 Complex

Two-dimensional stationary data stream containing 100 000 points describing a complex cluster structure, with seven clear clusters; used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models. It is also split into four quadrants when illustrating the distributed and collaborative learning methodologies — illustrated in Figure B.1b.

Name	$d$	$N$	Stationary	Change at	#Clusters	Chapters	Figure
<i>Gauss</i>	2	100 000	Y	-	1	4, 5, C	B.1a
<i>Complex</i>	2	100 000	Y	-	7	4, 5, 6	B.1b
<i>Chain</i>	3	100 000	Y	-	2	4, 5, C	B.1c
<i>d2k20</i>	2	300 000	Y	-	20	4, 5, C	B.1d
<i>d3k20</i>	3	300 000	Y	-	20	4, 5, C	
<i>d5k20</i>	5	300 000	Y	-	20	4, 5, C	
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2	4, 5, C	B.2a
<i>DriftTwoOne</i>	2	200 000	N	[50 001, 150 000]	2/1	4, 5, C	B.2b
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2	4, 5	B.3a
<i>Hepta</i>	3	150 000	N	100 001	7/6	4, 5	B.3b
<i>Correlated</i>	10	100 000	N	50 001	NA	5	B.4

Table B.1: Summary of artificial data streams, where  $d$  is the dimensionality of the data stream and  $N$  is the number of observations.

### B.3 Chain

Three-dimensional stationary data stream containing 100 000 points describing two interlocked rings (clusters); used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models — illustrated in Figure B.1c.

### B.4 d2k20, d3k20 and d5k20

Stationary data streams consisting of 300 000 points in  $d$  dimensions, describing 20 Gaussian clusters in different dimensions. Obtained from publicly disclosed data in (Frahling and Sohler 2008). Following the original description from the authors: Each individual data stream “is generated by taking a random point from one of 20 Gaussian distributed clusters, whose center are picked uniformly at random from the unit cube. The standard deviation of the Gaussian distribution is  $0.02\sqrt{d}$ , i.e., it is the product of the one-dimensional Gaussian distribution with standard deviation 0.02” — quoted from (Frahling and Sohler 2008). The data stream is used to evaluate if the clusters are detectable in the U-Matrix visualization of obtained SOM models; it is also used to test scalability of algorithm in Chapter 4 — the three-dimensional variant, i.e., *d3k20*, is illustrated in Figure B.1d.

### B.5 AbruptOneTwo

Two-dimensional non-stationary data stream containing 200 000 points describing an abrupt-changing (change point at  $t = 50\,001$ ) simple cluster structure with one Gaussian cluster splitting in two; aims at evaluating how different algorithms react to sudden

change in the distribution — illustrated in Figure B.2a.

## B.6 DriftTwoOne

Two-dimensional non-stationary data stream containing 200 000 points describing a gradual change in a simple cluster structure, i.e., two *Gaussian* clusters merging into one (change occurs during  $t = [50\,001, 150\,000]$ ); aims at evaluating how different algorithms react to gradual change in the underlying distribution — illustrated in Figure B.2b.

## B.7 Clouds

Two-dimensional non-stationary data stream containing 200 000 points describing a gradual changing cluster structure of three *Gaussian* clusters (change occurs during  $t = [50\,001, 150\,000]$ ). In the initial stationary phase of the stream there are only two separable Gaussian clusters (two are overlapped); during the non-stationary phase of the stream, the overlapped clusters split, i.e., one remains stationary, while the other drifts. In this phase, the third cluster also drifts, hence there are three separable clusters. Clusters move at different “speeds”, until they merge; in the final stationary phase of the data stream there are again only two separable clusters. Aims at evaluating how different algorithms react to gradual change in the underlying distribution — illustrated in Figure B.3a.

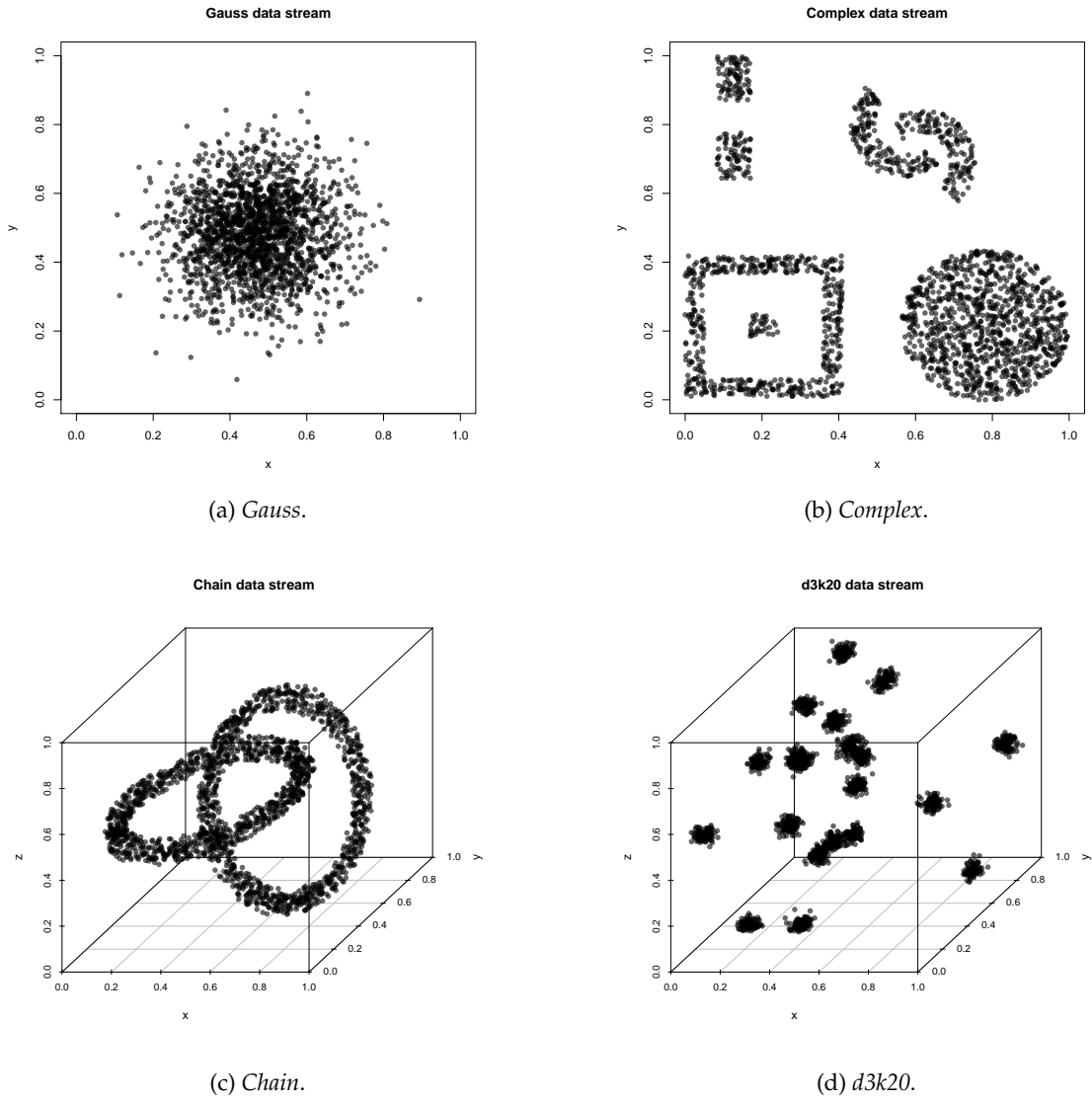
## B.8 Hepta

Three-dimensional non-stationary data stream containing 150 000 points describing an abrupt-changing cluster structure (change point at  $t = 100\,001$ ). From the initial seven *Gaussian* clusters (hence the name), one disappears; aims at evaluating how different algorithms react to sudden change in the underlying distribution and to motivate the proposal of the *average neuron utility* assessment metric in Chapter 5 — illustrated in Figure B.3b.

## B.9 Correlated

As opposed to the previous data streams, this is utilized to evaluate the feature clustering methodology, i.e., time-series clustering. The data stream contains 5 pairs of correlated time-series ( $d = 10$ ) with different degrees of correlation. The pairwise correlations abruptly change at  $t = 50\,001$ . Table B.2 depicts the correlations along time. Generated in *R* using the *rmvnorm* function; script is available in the repository provided in the beginning of the appendix. A small sample (100 observations) is depicted in Figure B.4.

Degree of Correlation	First half, $t \in [0, 50\,000]$	Second half, $t = [50\,001, 100\,000]$
0	{a,b}	{g,h}
0.4	{c,d}	{c,d}
0.8	{e,f}	{e,f}
1	{g,h}	{i,j}
-0.6	{i,j}	{a,b}

Table B.2: Pairwise correlations in the *Correlated* artificial data stream.Figure B.1: Illustration of several stationary artificial data streams (*Gauss*, *Complex*, *Chain* and *d3k20*).

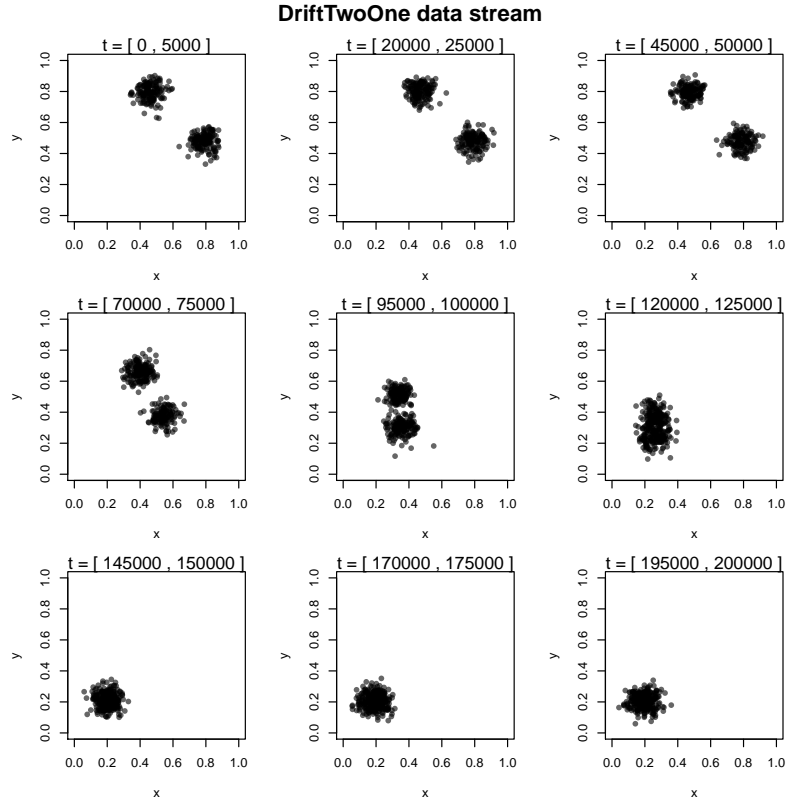
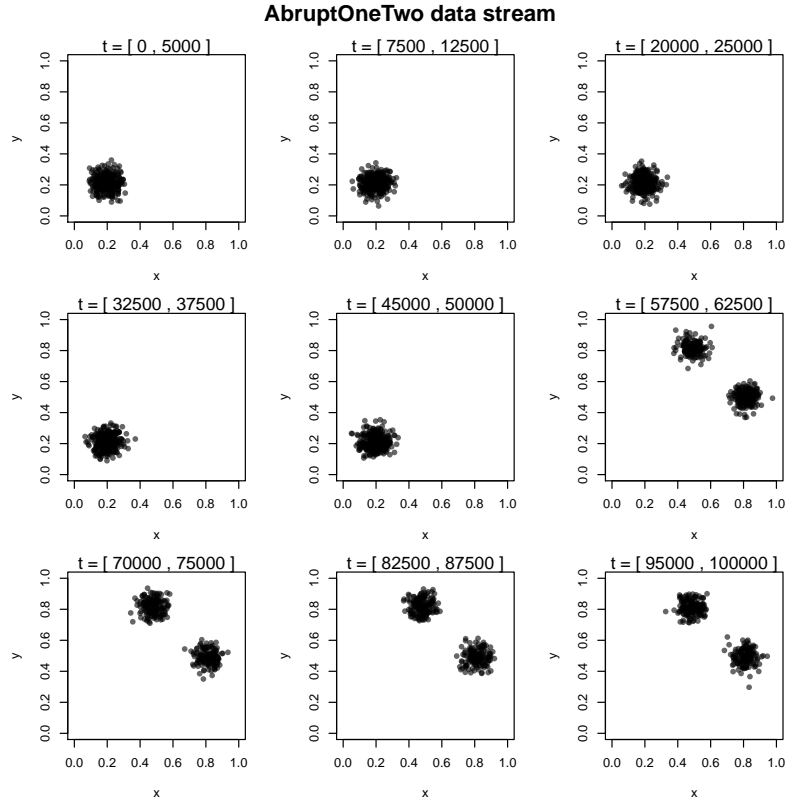


Figure B.2: Illustration of non-stationary artificial data streams (*AbruptOneTwo* and *DriftTwoOne*).

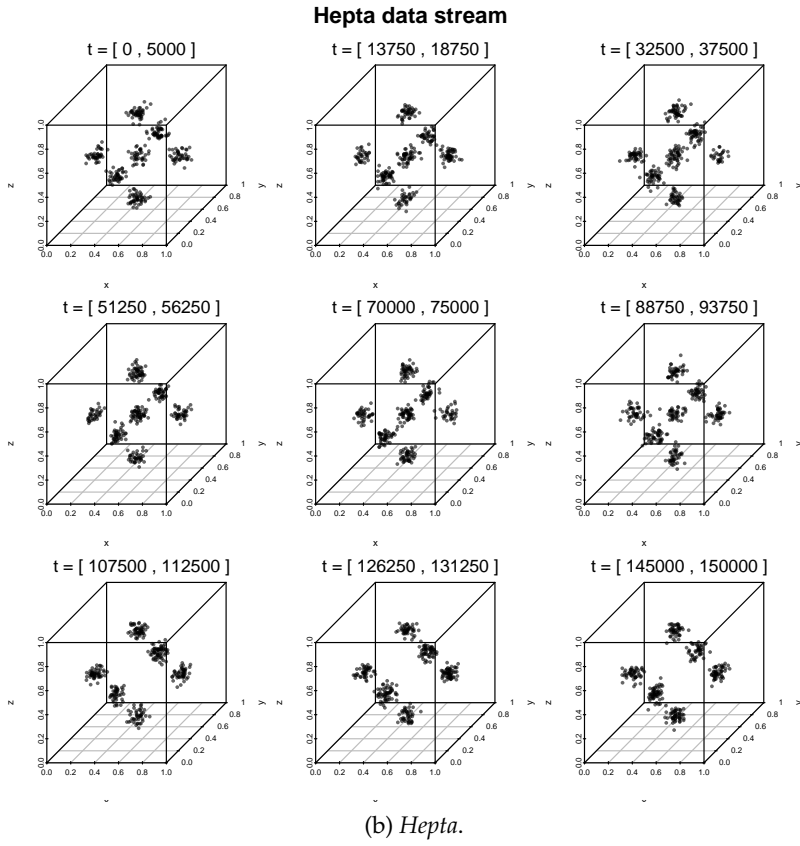
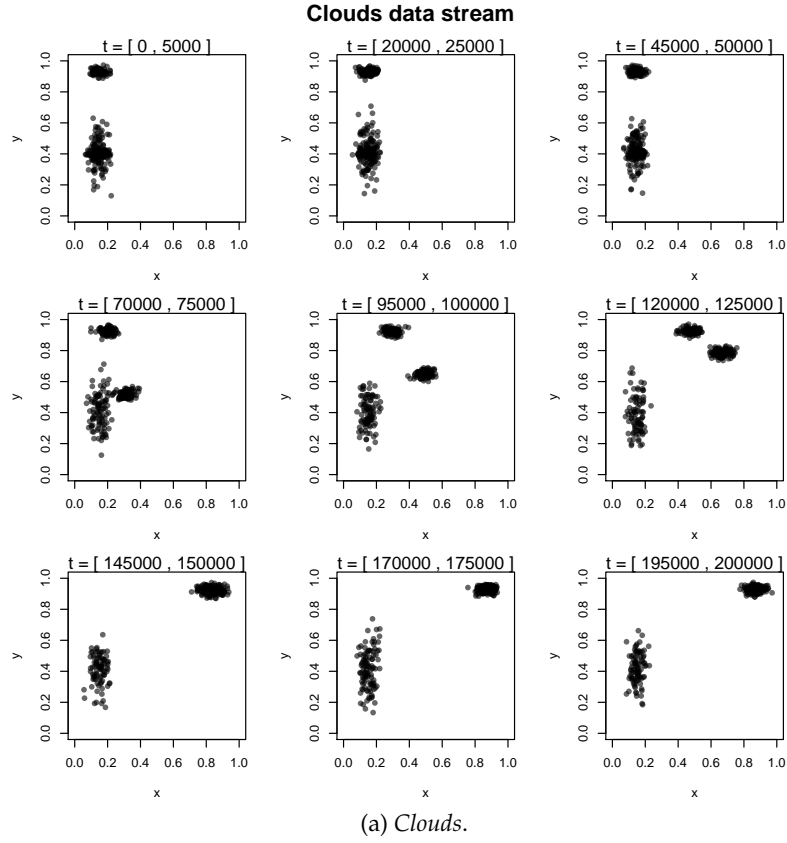


Figure B.3: Illustration of additional non-stationary artificial data streams (*Clouds* and *Hepta*).

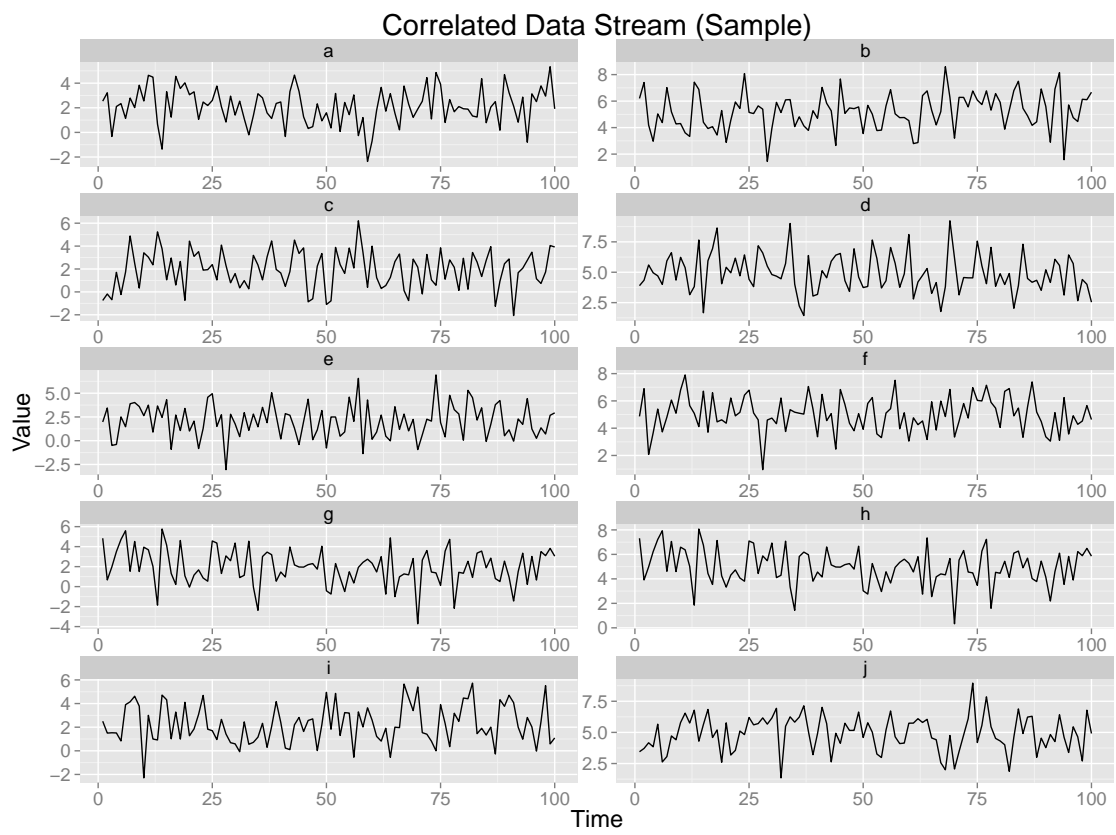


Figure B.4: Illustration of the non-stationary *Correlated* data stream.







# Additional Results from Experimental Evaluations

This appendix presents additional results from the experiments endured in Chapters 4 and 5, concerning the StreamART2A and Ubiquitous Self-Organizing Map (UbiSOM) algorithms.

## C.1 StreamART2A Algorithm

Table C.1 summarizes results contained in this section, in addition to results already presented in Section 4.5.

### C.1.1 Parameter Sensitivity Analysis

The parameter sensitivity analysis was performed in three dimensions with  $L = \{500, 1000, 1500, 2000\}$ ,  $q = \{20, 50, 100, 200\}$  and  $\eta = \{0.05, 0.1, 0.2, 0.5\}$  as the parameter-space, against the following two statistics:

**Mean quantization error.** After a landmark is processed and  $q$  micro-categories are generated, the presented  $L$  observations are reused to compute the mean quantization error of the landmark window, i.e.,  $\left( \sum_{i=t}^{t+L-1} E'_q(t) \right) / L$ , being  $E'_q(t)$  computed with  $w_c(t)$  inside the landmark window. The final obtained statistic (Mean  $E'_q(t)$ ) is the mean value obtained between all landmark windows until the end of the data stream.

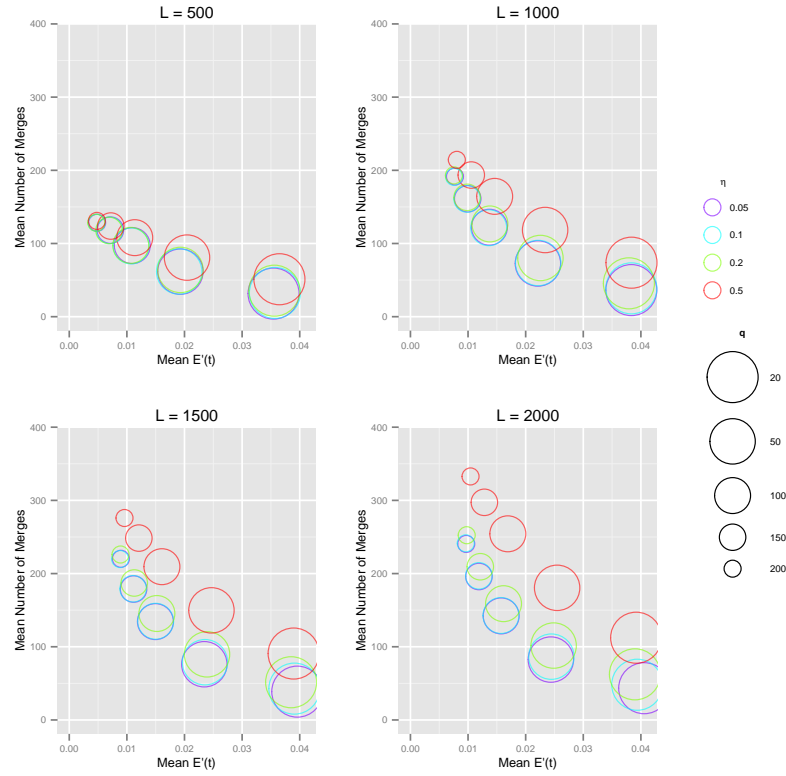
Name	$d$	$N$	Data Stream			Type of Evaluation			
			Stationary	Change at	#Clusters	PSA	VIL	ECA	MA
<i>Gauss</i>	2	100 000	Y	-	1	+			
<i>Complex</i>	2	100 000	Y	-	7				
<i>Chain</i>	3	100 000	Y	-	2	+			
<i>d2k20</i>	2	300 000	Y	-	20	+			
<i>d3k20</i>	3	300 000	Y	-	20	+			
<i>d5k20</i>	5	300 000	Y	-	20				
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2				
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2				
<i>Hepta</i>	3	150 000	N	100 001	7/6	+			

Table C.1: Additional results for the StreamART2A algorithm, marked with the symbol  $+$ , regarding: parameter sensitivity analysis (PSA); vigilance impact and landmark examples (VIL); exploratory cluster analysis (ECA), and; model assessment and change indication (MA).

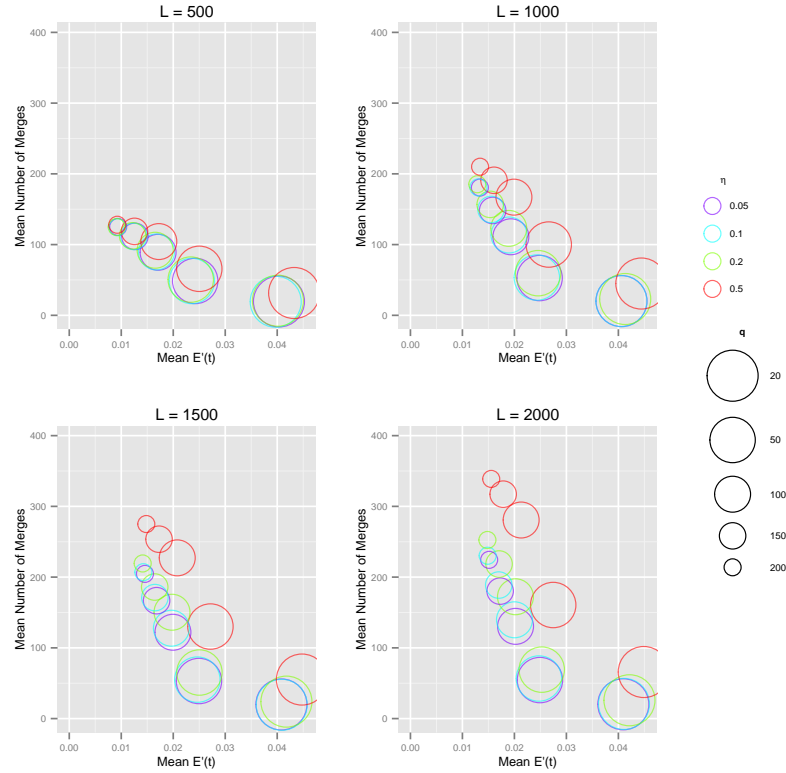
**Mean number of merges.** Computes the mean number of merges across all landmark windows, i.e., for each landmark window the number of micro-category merges is accumulated and averaged between the total number of landmark windows used throughout the data stream.

Figure C.1 presents the results of the PSA analysis for the *Gauss* and *Chain* stationary data streams; Analogously, Figure C.2 presents the results for the *d2k20* and *d3k20* data streams. Additionally, Figure C.3 presents the results for the non-stationary *Hepta* data stream.

Discussion of these results and regarding other data streams is made in Section 4.5.3.

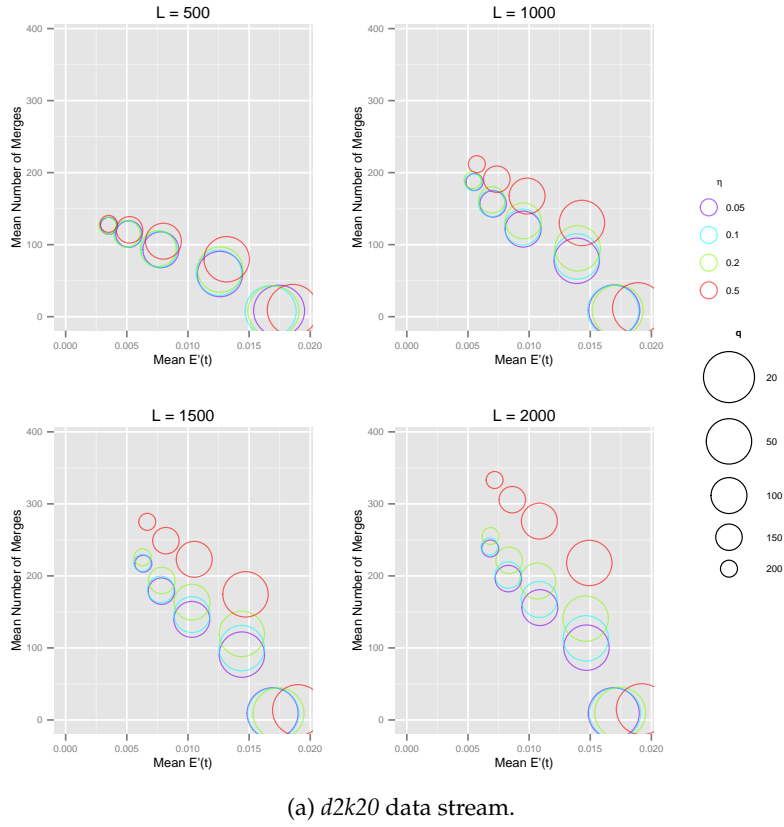


(a) *Gauss* data stream.

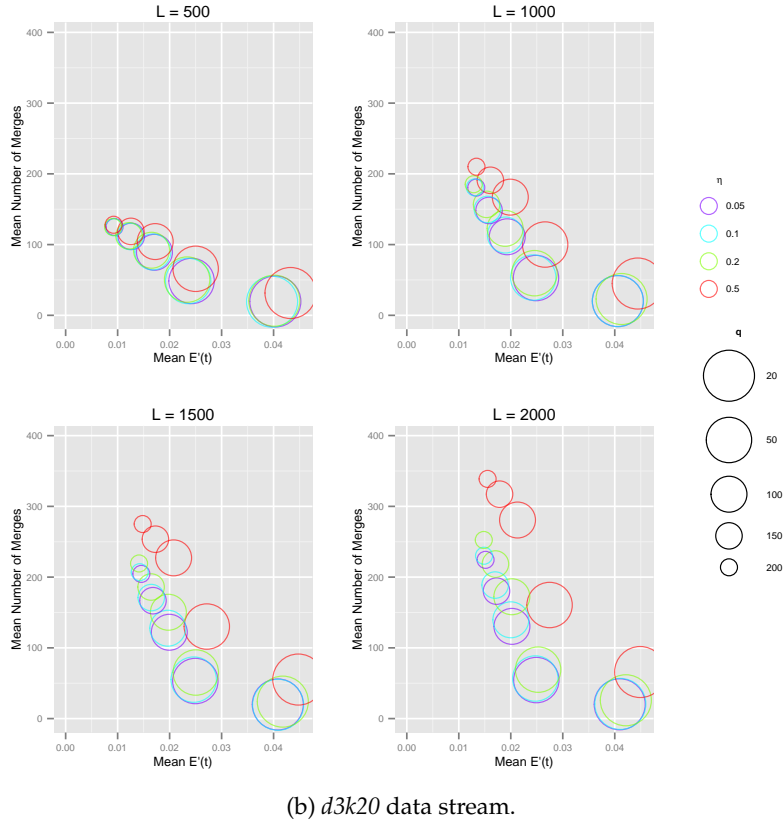


(b) *Chain* data stream.

Figure C.1: Additional parameter sensitivity analysis results for the StreamART2A algorithm over *Gauss* and *Chain* stationary data streams.

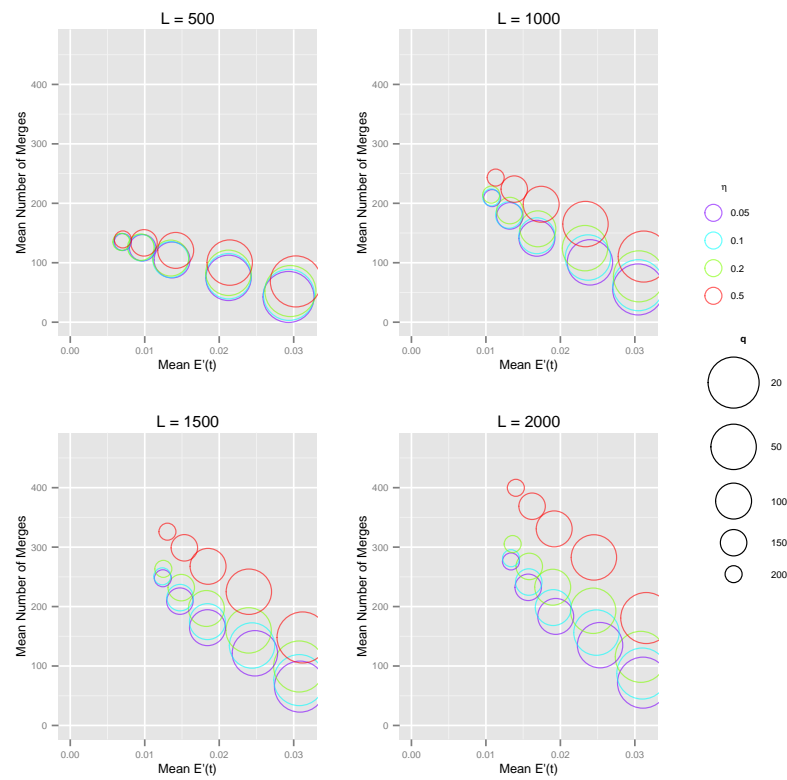


(a)  $d2k20$  data stream.



(b)  $d3k20$  data stream.

Figure C.2: Additional parameter sensitivity analysis results for the StreamART2A algorithm over  $d2k20$  and  $d3k20$  stationary data streams.

(a) *Hepta* data stream.Figure C.3: Additional parameter sensitivity analysis results for the StreamART2A algorithm over the non-stationary *Hepta* data stream.

## C.2 UbiSOM Algorithm

The following table summarizes the results contained in this section.

Name	$d$	$N$	Data Streams			Type of Evaluation		
			Stationary	Change at	#Clusters	PSA	LPC	ECA
<i>Gauss</i>	2	100 000	Y	-	1	+	+	
<i>Complex</i>	2	100 000	Y	-	7	+		
<i>Chain</i>	3	100 000	Y	-	2	+		
<i>d5k20</i>	5	300 000	Y	-	20	+	+	+
<i>AbruptOneTwo</i>	2	100 000	N	50 001	1/2	+	+	
<i>DriftTwoOne</i>	2	200 000	N	[50 001, 150 000]	2/1	+	+	
<i>Clouds</i>	2	200 000	N	[50 001, 150 000]	2/3/2	+		
<i>Hepta</i>	3	150 000	N	100 001	7/6	+		

Table C.2: Summary of additional experimental results for the UbiSOM algorithm, marked with the symbol  $+$ , regarding: parameter sensitivity analysis (PSA); evolution of learning parameters and convergence (LPC), and; exploratory cluster analysis (ECA).

### C.2.1 Description and Algorithm Parameters

A  $20 \times 40$  lattice was used in all runs. In the *ordering* state of the UbiSOM algorithm we have empirically set  $\eta_i = 0.1$ ,  $\eta_f = 0.08$ ,  $\sigma_i = 0.6$  and  $\sigma_f = 0.2$ , while parameters  $T$  and  $\beta$  vary in the parameter sensitivity analysis and in the other presented experiments.

### C.2.2 Parameter Sensitivity Analysis

The analysis was performed in two dimensions: sliding window size of assessment metrics  $T = \{500, 1000, 1500, 2000, 2500, 3000\}$  and drift function weighting factor  $\beta = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ . To assess the impact of these parameters, the following values were computed:

**Mean error and neuron activity.** The *mean quantization error* (Mean  $E'_q(t)$  or  $\overline{QE}$ ) characterizes the quality of the quantization procedure along the entire stream, i.e., we should look for lower values (see Section 2.4.6). Similarly, the *mean neuron activity* (Mean  $\lambda(t)$ ) characterizes the neuron usage during learning from stationary and non-stationary data streams, measured between 0 and 1. While it is normal to have some unused neurons separating clusters while projecting the input space, a large portion of unused neurons is undesirable. Consequently, we should look for higher values of this measure. Both measurements must not be confused with the assessment metrics  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$ , since they average values within a sliding window.

**Topographic error.** The *mean topographic error* (Mean  $TE(t)$  or  $\overline{TE}$ ) measures undesirable distortions of the map, i.e., topological defects (see Sections 2.4.4 and 2.4.6). Values greater than zero may not indicate topological defects, but should remain fairly close to zero.

**Convergence time.** The assessment metric  $\overline{qe}(t)$ , for the different values of  $T$ , was used to obtain a grasp on the delay in convergence imposed by this parameter. From the minimum  $\overline{qe}(t)$  obtained throughout the stream, iteration where the  $\overline{qe}(t)$  value falls within 5% of the minimum was computed (Convergence  $t$ ), as a temporal indicator of convergence. In other words, this value indicates at which point in time the map attained a convergence of 95% against the minimum  $\overline{qe}(t)$  obtained throughout the data stream. This value is merely indicative and should be regarded with care, as it is a hard comparison, i.e., a value very close can be obtained much earlier, so it is not one of the main quantitative results we are interested in.

**Number of resets.** The number of transitions back to the *ordering state* (resets) was also obtained. A transition to this state indicates that the UbiSOM algorithm was unable to adapt to the underlying distribution with the established learning parameters thresholds inherited from the ordering state, i.e.,  $\eta_f$  and  $\sigma_f$ . Ideally, during the presentation of the artificial data streams, we want the least number of resets; this indicates that the established parameters allowed the UbiSOM to properly adjust itself to any changes in the underlying data stream.

The objective functions  $f_{QE}(T, \beta)$ ,  $f_{\lambda}(T, \beta)$  and  $f_{TE}(T, \beta)$  represent values obtained for  $Mean E'_q(t)$ ,  $Mean \lambda(t)$  and  $Mean TE(t)$ , respectively for a particular parameterization. The best parameters for a particular data stream are obtained by maximization of the *optimization function*:

$$\underset{T, \beta}{maximize} \quad g(T, \beta) = \frac{f_{\lambda}(T, \beta)}{f_{\lambda}^{max}} - \frac{f_E(T, \beta)}{f_{QE}^{max}} - \frac{f_{TE}(T, \beta)}{f_{TE}^{max}}$$

with an upper limit of one and an unbounded lower limit (please note that  $f_i^{max} \neq 0$  is assumed).

Tables C.3 through C.10 present the full results for all combinations of  $T$  and  $\beta$  over all artificial data streams. Additionally, the optimization scores are also presented here. Discussion of the results can be found in Section 5.5.3.

### C.2.3 Convergence with stationary and non-stationary data

This section presents additional results regarding the ones presented in Section 5.5.4, namely for the *Gauss* (Figure C.4), *AbruptOneTwo* (Figure C.5) and *DriftTwoOne* (Figures C.6 and C.7) artificial data streams. The UbiSOM parameterization for each problem stems from the performed PSA. For each data stream the UbiSOM lattice is shown at

$T$	$\beta$	Mean $\hat{E}_q'(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	7.22154e-03	9.997360e-01	8.600e-03	10 381	0	-3.702832e-01
	0.6	7.20783e-03	9.997111e-01	8.700e-03	10 386	0	-3.742498e-01
	0.7	7.19959e-03	9.996822e-01	8.820e-03	10 390	0	-3.800166e-01
	0.8	7.19434e-03	9.996685e-01	8.930e-03	10 390	0	-3.855726e-01
	0.9	7.19049e-03	9.996494e-01	8.980e-03	10 394	0	-3.879314e-01
	1.0	7.18898e-03	9.996320e-01	8.950e-03	10 391	0	-3.860760e-01
	0.5	7.61592e-03	9.999959e-01	1.776e-02	10 569	0	-9.336128e-01
	0.6	7.61552e-03	9.999959e-01	1.778e-02	10 569	0	-9.346886e-01
	0.7	7.61553e-03	9.999959e-01	1.777e-02	10 568	0	-9.341274e-01
	0.8	7.61520e-03	9.999959e-01	1.775e-02	10 569	0	-9.329620e-01
1 000	0.9	7.61528e-03	9.999959e-01	1.773e-02	10 568	0	-9.318470e-01
	1.0	7.61523e-03	9.999959e-01	1.769e-02	10 568	0	-9.295911e-01
	0.5	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	0.6	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	0.7	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	0.8	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	0.9	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	1.0	7.50597e-03	1	7.960e-03	14 314	0	-3.689331e-01
	0.5	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
	0.6	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
2 000	0.7	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
	0.8	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
	0.9	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
	1.0	7.3615e-03	1	7.760e-03	14 621	0	-3.859354e-01
	0.5	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
	0.6	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
	0.7	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
	0.8	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
	0.9	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
	1.0	7.93710e-03	1	7.940e-03	14 900	0	-4.207226e-01
3 000	0.5	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01
	0.6	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01
	0.7	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01
	0.8	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01
	0.9	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01
	1.0	8.14769e-03	1	7.800e-03	15 312	0	-4.386952e-01

Table C.3: UbiSOM parameter sensitivity analysis for *Gauss* data stream.

$T$	$\beta$	Mean $\hat{E}_q'(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	1.226954e-02	9.879168e-01	2.242e-02	4 078	0	-8.908477e-01
	0.6	1.014159e-02	9.855844e-01	2.546e-02	5 140	0	-8.391539e-01
	0.7	9.527250e-03	9.815950e-01	2.281e-02	5 186	0	-6.889958e-01
	0.8	9.172180e-03	9.744567e-01	1.764e-02	6 213	0	-6.641442e-01
	0.9	9.046870e-03	9.585747e-01	1.278e-02	6 301	0	-2.789550e-01
	1.0	9.569450e-03	8.253973e-01	8.600e-03	6 331	0	-2.907919e-01
	0.5	9.630880e-03	9.935557e-01	2.013e-02	6 404	0	-5.801956e-01
	0.6	9.433190e-03	9.911198e-01	1.710e-02	6 486	0	-4.475135e-01
	0.7	9.211560e-03	9.871500e-01	1.516e-02	6 543	0	-3.572293e-01
	0.8	9.227960e-03	9.812221e-01	1.580e-02	6 582	0	-3.896423e-01
1 000	0.9	9.349870e-03	9.693131e-01	1.551e-02	6 681	0	-4.001189e-01
	1.0	9.664000e-03	9.084136e-01	1.391e-02	6 588	0	-4.238901e-01
	0.5	9.384440e-03	9.973164e-01	7.940e-03	9 299	0	-7.755208e-02
	0.6	9.375210e-03	9.964025e-01	7.750e-03	9 285	0	-7.025276e-02
	0.7	9.385980e-03	9.947991e-01	7.490e-03	9 267	0	-6.252476e-02
	0.8	9.404490e-03	9.919812e-01	6.890e-03	9 209	0	-4.329017e-02
	0.9	9.436820e-03	9.842766e-01	6.950e-03	9 197	0	-5.600068e-02
	1.0	9.539140e-03	9.624654e-01	8.900e-03	9 110	0	-1.627824e-01
	0.5	9.707880e-03	9.976559e-01	7.340e-03	9 505	0	-8.000684e-02
	0.6	9.692260e-03	9.965985e-01	8.140e-03	9 495	0	-1.112149e-01
2 000	0.7	9.713740e-03	9.952287e-01	7.320e-03	9 440	0	-8.213057e-02
	0.8	9.733270e-03	9.928904e-01	7.290e-03	9 390	0	-8.488665e-02
	0.9	9.759050e-03	9.864683e-01	7.580e-03	9 362	0	-1.048122e-01
	1.0	9.856750e-03	9.690873e-01	8.660e-03	9 195	0	-1.726078e-01
	0.5	1.012475e-02	9.977844e-01	6.540e-03	9 850	0	-8.243226e-02
	0.6	1.012334e-02	9.970303e-01	6.380e-03	9 926	0	-7.678845e-02
	0.7	1.014682e-02	9.957217e-01	5.930e-03	9 929	0	-6.233836e-02
	0.8	1.016753e-02	9.934998e-01	5.750e-03	9 855	0	-5.918237e-02
	0.9	1.019101e-02	9.873335e-01	5.660e-03	9 800	0	-6.373884e-02
	1.0	1.026647e-02	9.771283e-01	6.730e-03	9 661	0	-1.221399e-01
3 000	0.5	1.055319e-02	9.981485e-01	6.270e-03	10 487	0	-1.063816e-01
	0.6	1.055985e-02	9.974771e-01	6.050e-03	10 483	0	-9.895603e-02
	0.7	1.057594e-02	9.961976e-01	5.620e-03	10 426	0	-8.466006e-02
	0.8	1.059429e-02	9.942452e-01	5.530e-03	10 408	0	-8.457666e-02
	0.9	1.061370e-02	9.905050e-01	5.420e-03	10 340	0	-8.553969e-02
	1.0	1.069329e-02	9.814270e-01	5.200e-03	10 232	0	-9.252597e-02

Table C.4: UbiSOM parameter sensitivity analysis for *Complex* data stream.



$T$	$\beta$	Mean $E_q'(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	3.659760e-02	9.677948e-01	4.300000e-04	2 119	0	2.117215e-01
	0.6	3.530797e-02	9.538220e-01	9.633300e-04	2 317	0	2.217200e-01
	0.7	2.622263e-02	9.325730e-01	1.966670e-03	2 552	0	3.850209e-01
	0.8	1.467811e-02	9.465237e-01	4.962333e-02	3 122	0	3.865103e-01
	0.9	1.322241e-02	9.263288e-01	9.283667e-02	3 318	0	1.659520e-01
	1.0	1.437058e-02	4.313674e-01	1.510067e-01	3 679	0	-6.697049e-01
1 000	0.5	4.777409e-02	9.649294e-01	1.410000e-03	3 368	0	-3.035390e-02
	0.6	3.405412e-02	9.505531e-01	1.360000e-03	3 409	0	2.425386e-01
	0.7	2.258051e-02	9.359845e-01	6.583330e-03	3 559	0	4.400783e-01
	0.8	1.366186e-02	9.694970e-01	1.102200e-01	4 650	0	1.077153e-01
	0.9	1.362352e-02	9.287341e-01	9.737667e-02	4 638	0	1.357673e-01
	1.0	1.468751e-02	4.657999e-01	1.518833e-01	4 331	0	-6.461472e-01
1 500	0.5	4.560235e-02	9.670192e-01	3.046670e-03	4 469	0	8.488053e-03
	0.6	3.159775e-02	9.540288e-01	2.893330e-03	4 481	0	2.892931e-01
	0.7	2.179408e-02	9.379426e-01	4.513330e-03	4 530	0	4.695678e-01
	0.8	1.358589e-02	9.792678e-01	1.253600e-01	5 226	0	3.841621e-02
	0.9	1.369168e-02	9.486619e-01	1.134967e-01	5 184	0	6.850807e-02
	1.0	1.460136e-02	5.615483e-01	1.748433e-01	4 975	0	-6.698909e-01
2 000	0.5	4.384042e-02	9.684831e-01	5.653330e-03	5 796	0	3.294234e-02
	0.6	3.036333e-02	9.559315e-01	6.543330e-03	5 806	0	2.975830e-01
	0.7	2.025618e-02	9.416751e-01	8.580000e-03	5 842	0	4.838398e-01
	0.8	1.402405e-02	9.758631e-01	1.153100e-01	6 392	0	7.942166e-02
	0.9	1.414113e-02	9.467546e-01	1.090167e-01	6 379	0	8.107310e-02
	1.0	1.485825e-02	6.112792e-01	1.800833e-01	6 212	0	-6.528659e-01
2 500	0.5	4.249120e-02	9.690999e-01	7.916670e-03	6 971	0	4.973187e-02
	0.6	2.867538e-02	9.570476e-01	9.173330e-03	6 979	0	3.200121e-01
	0.7	1.481907e-02	9.851281e-01	1.106733e-01	7 423	0	9.690320e-02
	0.8	1.433585e-02	9.816028e-01	1.265567e-01	7 429	0	1.869761e-02
	0.9	1.444161e-02	9.563099e-01	1.179033e-01	7 407	0	3.704260e-02
	1.0	1.510030e-02	6.483013e-01	1.826067e-01	7 290	0	-6.339048e-01
3 000	0.5	3.670820e-02	9.681722e-01	9.090000e-03	7 861	0	1.635806e-01
	0.6	2.568350e-02	9.572635e-01	8.023330e-03	7 855	0	3.889924e-01
	0.7	1.476446e-02	9.874738e-01	1.255033e-01	8 346	0	2.129193e-02
	0.8	1.458755e-02	9.820777e-01	1.311633e-01	8 336	0	-1.067029e-02
	0.9	1.469716e-02	9.579913e-01	1.201233e-01	8 310	0	2.155068e-02
	1.0	1.524109e-02	6.705210e-01	1.874133e-01	8 192	0	-6.399976e-01

Table C.6: UbiSOM parameter sensitivity analysis for  $d5k20$  data stream.Table C.5: UbiSOM parameter sensitivity analysis for  $Chain$  data stream.

$T$	$\beta$	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	4.38363e-03	9.997364e-01	8.89e-03	7 097	0	-7.078195e-01
	0.6	4.35046e-03	9.996733e-01	9.46e-03	7 123	0	-7.587525e-01
	0.7	4.32545e-03	9.996407e-01	9.67e-03	8 260	0	-7.751411e-01
	0.8	4.31921e-03	9.995747e-01	9.91e-03	8 264	0	-7.980672e-01
	0.9	4.31551e-03	9.994804e-01	9.79e-03	8 266	0	-7.854593e-01
	1.0	4.31374e-03	9.993422e-01	9.99e-03	8 267	0	-8.052874e-01
1 000	0.5	4.77001e-03	9.999377e-01	7.12e-03	8 623	1	-6.025113e-01
	0.6	4.76664e-03	9.999408e-01	7.34e-03	8 623	1	-6.239016e-01
	0.7	4.76411e-03	9.999377e-01	7.42e-03	8 623	1	-6.314409e-01
	0.8	4.76512e-03	9.999351e-01	7.53e-03	8 623	1	-6.426428e-01
	0.9	4.76282e-03	9.999354e-01	7.44e-03	8 623	1	-6.332046e-01
	1.0	4.76206e-03	9.999270e-01	7.48e-03	8 623	1	-6.370752e-01
1 500	0.5	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
	0.6	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
	0.7	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
	0.8	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
	0.9	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
	1.0	4.73490e-03	1	6.89e-03	8 906	0	-5.728771e-01
2 000	0.5	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
	0.6	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
	0.7	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
	0.8	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
	0.9	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
	1.0	4.99266e-03	1	7.47e-03	12 519	0	-6.790144e-01
2 500	0.5	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
	0.6	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
	0.7	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
	0.8	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
	0.9	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
	1.0	5.21247e-03	1	6.08e-03	16 041	0	-5.808758e-01
3 000	0.5	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01
	0.6	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01
	0.7	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01
	0.8	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01
	0.9	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01
	1.0	5.36115e-03	1	5.35e-03	13 091	0	-5.353355e-01

Table C.7: UbiSOM parameter sensitivity analysis for *AbrruptOneTwo* data stream.

$T$	$\beta$	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	5.22414e-03	9.985755e-01	1.5330e-02	21 387	1	-8.149456e-01
	0.6	5.14299e-03	9.972757e-01	1.6430e-02	15 791	0	-8.647634e-01
	0.7	5.09008e-03	9.953763e-01	1.7445e-02	21 360	0	-9.153686e-01
	0.8	5.04623e-03	9.910925e-01	1.7225e-02	15 795	0	-8.991903e-01
	0.9	4.94304e-03	9.795986e-01	1.1970e-02	15 786	0	-5.909780e-01
	1.0	5.53565e-03	7.347355e-01	1.3185e-02	5 961	0	-1.011956e+00
1 000	0.5	4.94659e-03	9.987274e-01	5.2350e-03	29 524	0	-1.863917e-01
	0.6	4.86550e-03	9.982649e-01	6.8600e-03	29 521	0	-2.654765e-01
	0.7	4.89365e-03	9.958548e-01	4.0800e-03	25 809	0	-1.135749e-01
	0.8	4.85912e-03	9.927826e-01	4.0850e-03	27 884	0	-1.107509e-01
	0.9	4.80230e-03	9.821779e-01	4.7600e-03	19 819	0	-1.498815e-01
	1.0	5.35486e-03	7.433131e-01	9.3700e-03	8 191	0	-7.522899e-01
1 500	0.5	5.05546e-03	9.986921e-01	5.4750e-03	27 684	0	-2.196898e-01
	0.6	4.96472e-03	9.980608e-01	5.5450e-03	20 070	0	-2.080774e-01
	0.7	4.79992e-03	9.972050e-01	6.9600e-03	33 870	0	-2.605206e-01
	0.8	4.82665e-03	9.929180e-01	4.3900e-03	26 214	0	-1.222816e-01
	0.9	4.88594e-03	9.823304e-01	4.8650e-03	19 936	0	-1.707328e-01
	1.0	5.44228e-03	7.527134e-01	9.2000e-03	11 921	0	-7.487956e-01
2 000	0.5	5.03535e-03	9.987983e-01	6.0050e-03	27 948	0	-2.463617e-01
	0.6	5.07621e-03	9.981944e-01	5.4900e-03	42 926	0	-2.247655e-01
	0.7	5.04500e-03	9.972478e-01	5.9650e-03	34 117	0	-2.473500e-01
	0.8	4.96230e-03	9.934211e-01	5.0600e-03	28 108	0	-1.844875e-01
	0.9	4.95394e-03	9.836751e-01	4.9100e-03	27 819	0	-1.841489e-01
	1.0	5.48447e-03	7.636120e-01	9.9600e-03	12 353	0	-7.890081e-01
2 500	0.5	5.48435e-03	9.985640e-01	5.4500e-03	46 222	1	-2.952252e-01
	0.6	5.17897e-03	9.979710e-01	5.3650e-03	46 135	0	-2.362344e-01
	0.7	5.11334e-03	9.972803e-01	5.8150e-03	46 185	0	-2.509629e-01
	0.8	4.95060e-03	9.955320e-01	6.5150e-03	34 332	0	-2.636828e-01
	0.9	5.03696e-03	9.843045e-01	5.3050e-03	26 390	0	-2.210353e-01
	1.0	5.53600e-03	7.726926e-01	9.7850e-03	16 305	0	-7.791172e-01
3 000	0.5	5.24434e-03	9.986762e-01	6.3500e-03	43 743	0	-3.037032e-01
	0.6	5.20917e-03	9.982327e-01	6.0900e-03	43 635	0	-2.829421e-01
	0.7	5.27926e-03	9.967641e-01	5.8550e-03	25 276	0	-2.834990e-01
	0.8	5.05163e-03	9.945718e-01	5.7150e-03	43 450	0	-2.368844e-01
	0.9	5.04961e-03	9.845118e-01	5.4100e-03	27 943	0	-2.291131e-01
	1.0	5.58158e-03	7.743236e-01	9.3450e-03	16 523	0	-7.604284e-01

Table C.8: UbiSOM parameter sensitivity analysis for *DriftTwoOne* data stream.

$T$	$\beta$	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score	
500	0.5	5.65335e-03	9.974773e-01	1.969549e-02	6 093	1	-5.637672e-01	
	0.6	5.33617e-03	9.966478e-01	2.208055e-02	6 939	1	-5.765969e-01	
	0.7	5.04886e-03	9.948301e-01	2.207555e-02	6 913	1	-5.274527e-01	
	0.8	5.20615e-03	9.924674e-01	2.692567e-02	6 896	0	-6.961328e-01	
	0.9	4.95146e-03	9.835369e-01	2.806070e-02	6 934	0	-6.924325e-01	
	1.0	4.60262e-03	9.838833e-01	3.502088e-02	6 905	0	-9.192280e-01	
	1 000	0.5	5.28792e-03	9.981059e-01	1.484037e-02	7 163	1	-3.598633e-01
		0.6	5.00901e-03	9.976649e-01	1.548039e-02	7 161	1	-3.292449e-01
		0.7	5.14164e-03	9.969742e-01	2.079052e-02	7 164	0	-5.052024e-01
		0.8	5.03380e-03	9.945676e-01	2.032051e-02	7 178	0	-7.49373e-01
0.9		4.88337e-03	9.879514e-01	2.139553e-02	7 186	0	-8.856489e-01	
1.0		4.57762e-03	9.315735e-01	2.748569e-02	7 201	0	-6.619095e-01	
1 500		0.5	5.20280e-03	9.985426e-01	1.516038e-02	9 457	1	-3.535073e-01
		0.6	5.18767e-03	9.982755e-01	1.711543e-02	9 466	1	-4.069237e-01
		0.7	5.09198e-03	9.971904e-01	2.098052e-02	9 471	0	-5.014491e-01
		0.8	5.03368e-03	9.947042e-01	2.058551e-02	9 487	0	-4.823464e-01
	0.9	4.88907e-03	9.884531e-01	2.170554e-02	9 496	0	-4.950069e-01	
	1.0	4.63415e-03	9.350103e-01	2.851571e-02	9 495	0	-6.978797e-01	
	2 000	0.5	5.23318e-03	9.988502e-01	1.400535e-02	9 794	1	-3.255920e-01
		0.6	5.27834e-03	9.982176e-01	1.515038e-02	9 794	1	-3.669092e-01
		0.7	5.34787e-03	9.971396e-01	1.610040e-02	9 794	1	-4.074145e-01
		0.8	5.10344e-03	9.957539e-01	2.141554e-02	9 794	0	-5.173361e-01
0.9		4.97764e-03	9.883764e-01	1.939548e-02	9 794	0	-4.447882e-01	
1.0		4.67874e-03	9.042697e-01	2.011550e-02	9 794	0	-4.968801e-01	
2 500		0.5	5.27584e-03	9.986992e-01	1.396535e-02	10 176	1	-3.321469e-01
		0.6	5.26397e-03	9.981086e-01	1.243031e-02	10 176	1	-2.868046e-01
		0.7	5.22708e-03	9.980996e-01	2.009050e-02	10 176	0	-4.990222e-01
		0.8	5.14383e-03	9.970674e-01	1.831046e-02	10 176	0	-4.345019e-01
	0.9	5.17867e-03	9.920275e-01	1.935048e-02	10 176	0	-4.754074e-01	
	1.0	4.90830e-03	9.512118e-01	1.695542e-02	10 176	0	-4.000559e-01	
	3 000	0.5	5.60788e-03	9.981644e-01	1.309533e-02	10 725	1	-3.665728e-01
		0.6	5.42420e-03	9.984614e-01	1.566539e-02	11 830	1	-4.071715e-01
		0.7	5.14219e-03	9.984398e-01	1.947049e-02	12 017	0	-4.659617e-01
		0.8	5.25701e-03	9.964523e-01	1.946549e-02	10 725	0	-4.881188e-01
0.9		5.20558e-03	9.922907e-01	1.987550e-02	10 725	0	-4.948956e-01	

Table C.10: UbiSOM parameter sensitivity analysis for *Hepta* data stream.

$T$	$\beta$	Mean $E'_q(t)$	Mean $\lambda(t)$	Mean TE(t)	Convergence $t$	Resets	Opt. Score
500	0.5	5.65335e-03	9.974773e-01	1.969549e-02	6 093	1	-5.637672e-01
	0.6	5.33617e-03	9.966478e-01	2.208055e-02	6 939	1	-5.765969e-01
	0.7	5.04886e-03	9.948301e-01	2.207555e-02	6 913	1	-5.274527e-01
	0.8	5.20615e-03	9.924674e-01	2.692567e-02	6 896	0	-6.961328e-01
	0.9	4.95146e-03	9.835369e-01	2.806070e-02	6 934	0	-6.924325e-01
	1.0	4.60262e-03	8.938833e-01	3.502088e-02	6 905	0	-9.192280e-01
1 000	0.5	5.28792e-03	9.981059e-01	1.484037e-02	7 163	1	-3.598633e-01
	0.6	5.00901e-03	9.976649e-01	1.548039e-02	7 161	1	-3.292449e-01
	0.7	5.14164e-03	9.969742e-01	2.079052e-02	7 164	0	-5.050244e-01
	0.8	5.03380e-03	9.945676e-01	2.032051e-02	7 178	0	-4.749375e-01
	0.9	4.88337e-03	9.879514e-01	2.139553e-02	7 186	0	-4.856489e-01
	1.0	4.57762e-03	9.315735e-01	2.748569e-02	7 201	0	-6.619095e-01
1 500	0.5	5.20280e-03	9.985426e-01	1.516038e-02	9 457	1	-3.535073e-01
	0.6	5.18767e-03	9.982755e-01	1.711543e-02	9 466	1	-4.069237e-01
	0.7	5.09198e-03	9.971904e-01	2.098052e-02	9 471	0	-5.014491e-01
	0.8	5.03368e-03	9.947042e-01	2.058551e-02	9 487	0	-4.823464e-01
	0.9	4.88907e-03	9.884531e-01	2.170554e-02	9 496	0	-4.950069e-01
	1.0	4.63415e-03	9.350103e-01	2.851571e-02	9 495	0	-6.978797e-01
2 000	0.5	5.23318e-03	9.988502e-01	1.400535e-02	9 794	1	-3.255920e-01
	0.6	5.27834e-03	9.982176e-01	1.515038e-02	9 794	1	-3.669092e-01
	0.7	5.34787e-03	9.971396e-01	1.610040e-02	9 794	1	-4.074145e-01
	0.8	5.10344e-03	9.957539e-01	2.141554e-02	9 794	0	-5.173361e-01
	0.9	4.97764e-03	9.883764e-01	1.939548e-02	9 794	0	-4.447882e-01
	1.0	4.67874e-03	9.042697e-01	2.011550e-02	9 794	0	-4.966801e-01
2 500	0.5	5.27584e-03	9.986992e-01	1.396535e-02	10 176	1	-3.321469e-01
	0.6	5.26397e-03	9.981086e-01	1.243031e-02	10 176	1	-2.868064e-01
	0.7	5.22708e-03	9.980996e-01	2.009050e-02	10 176	0	-4.990222e-01
	0.8	5.14383e-03	9.970674e-01	1.831046e-02	10 176	0	-4.345019e-01
	0.9	5.17867e-03	9.920275e-01	1.935048e-02	10 176	0	-4.754074e-01
	1.0	4.90830e-03	9.512118e-01	1.695542e-02	10 176	0	-4.000559e-01
3 000	0.5	5.67088e-03	9.981644e-01	1.309533e-02	10 725	1	-3.665728e-01
	0.6	5.42420e-03	9.984614e-01	1.566539e-02	11 830	1	-4.071715e-01
	0.7	5.14219e-03	9.984398e-01	1.947049e-02	12 017	0	-4.659617e-01
	0.8	5.25701e-03	9.964523e-01	1.946549e-02	10 725	0	-4.881188e-01
	0.9	5.20558e-03	9.922907e-01	1.987550e-02	10 725	0	-4.948956e-01
	1.0	4.96574e-03	9.539198e-01	1.766044e-02	10 725	0	-4.276365e-01

Table C.9: UbiSOM parameter sensitivity analysis for *Clouds* data stream.

specific stages of the data stream, *namely* (A) when the algorithm transitions to the *learning state*; (B) at *Convergence*  $t$  also from the PSA; (C) at the middle of the data stream, and; (D) at the end of the data stream, depicting how the map evolves over time. Also, obtained values for the local quantization error  $E'_q(t)$ , assessment metrics  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$ , and learning parameters  $\eta(t)$  and  $\sigma(t)$  are shown, depicting the evolution of the learning procedure. The drift function  $d(t)$  is not explicitly shown, since the learning parameters are estimated proportionally to it. Hence, the behavior of  $\eta(t)$  and  $\sigma(t)$  is the same as the drift function  $d(t)$  in the *learning state*, i.e., a weighted average between  $\overline{qe}(t)$  and  $\overline{\lambda}(t)$  at specific points in time. Regarding these results, the following can be observed:

**Gauss** (Figure C.4) — first of all, the UbiSOM replicates the quantization performed by the *Online SOM* algorithm in a streaming setting. After the initial unfolding by *ordering state* (A) the algorithm then proceeds to the *learning state* and evolves into a increasing better representation of the underlying distribution, consequence of the decreasing trend of the  $\overline{qe}(t)$  metric that reflects itself in the learning parameters. In this data stream the  $\overline{\lambda}(t)$  is always equal to 1 because all neurons are utilized in the quantization procedure.

**AbruptOneTwo** (Figure C.5) — after converging to the initial underlying distribution of the data stream, i.e., a single Gaussian cloud, an abrupt change occurs which is then reflected in the  $\overline{qe}(t)$  metric causing the learning parameters to increase significantly. Note that the algorithm was able to adapt to the new distribution, i.e., two Gaussian clouds, in less than  $T$  iterations. This fact is easily observed because the algorithm did not transition back to the *ordering state*. During both stable phases of the data stream the learning parameters decreased monotonically and allowed the algorithm to achieve a correct quantization of the input space.

**DriftTwoOne** (Figures C.6 and C.7) — these results pertain the discussion made at the end of Section 5.5.4, regarding a mismatch between the PSA obtained parameterization and a better one obtained experimentally. By using a relatively low  $\beta$  value (Figure C.7), although increasing the responsiveness to the moving clusters, this causes the lattice to perform “jumps” over this evolving distribution harming the quantization procedure, and therefore, the quantization errors. This is easily observed by comparing the behaviors of the  $\overline{\lambda}(t)$  metric along time, which causes spikes in the learning parameters.

## C.2.4 Exploratory Cluster Analysis

This section only presents an additional result for the *d5k20* stationary data stream that shows that a high number of clusters can be identified through the UbiSOM algorithm. Figure C.8 presents the final lattice and corresponding U-Matrix, together with the evolution of the assessment metrics and learning parameters. The evolution behavior is similar to, e.g., the *Complex* data stream and from the U-Matrix the expected 20 clusters can be inferred.

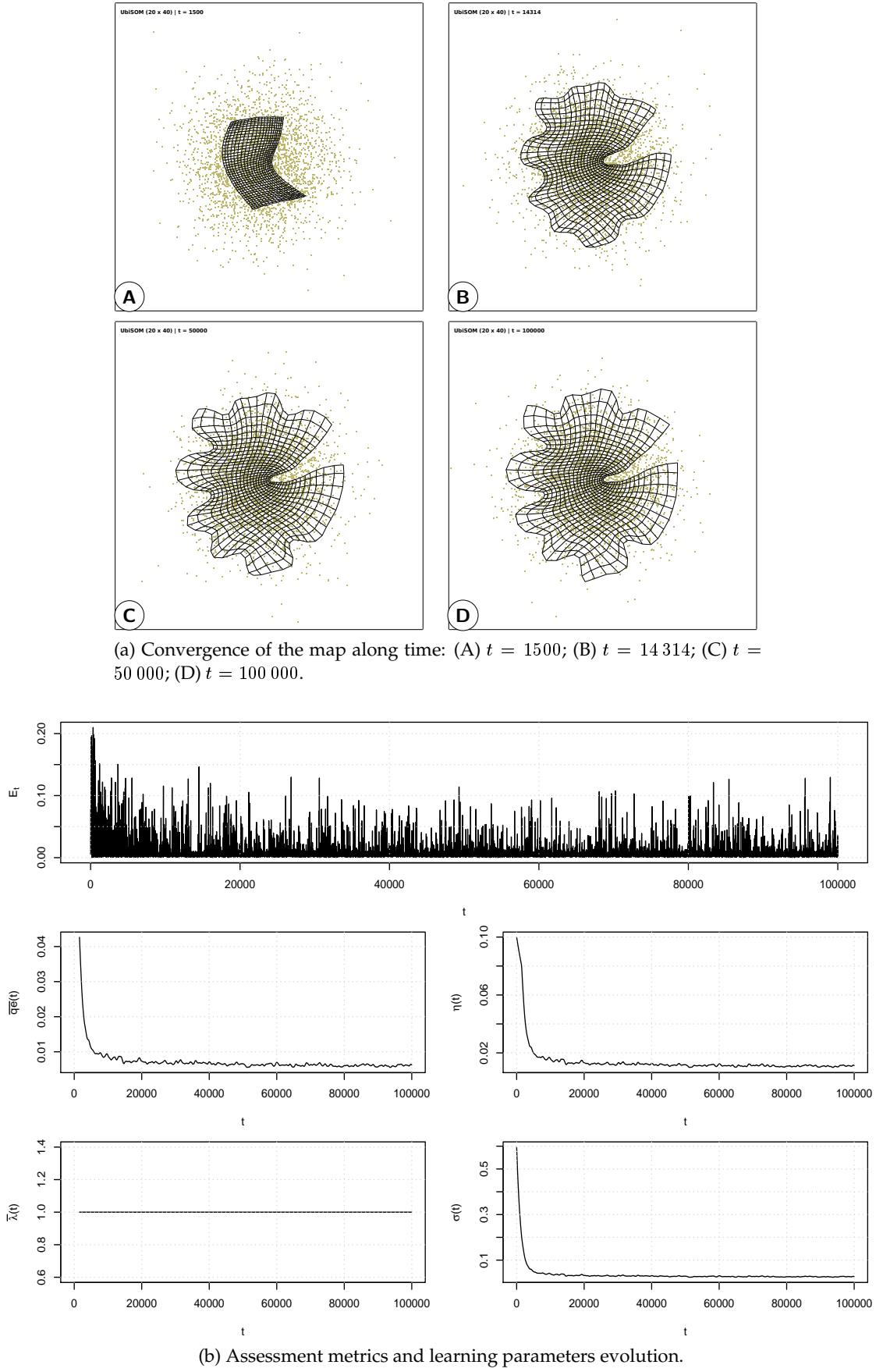
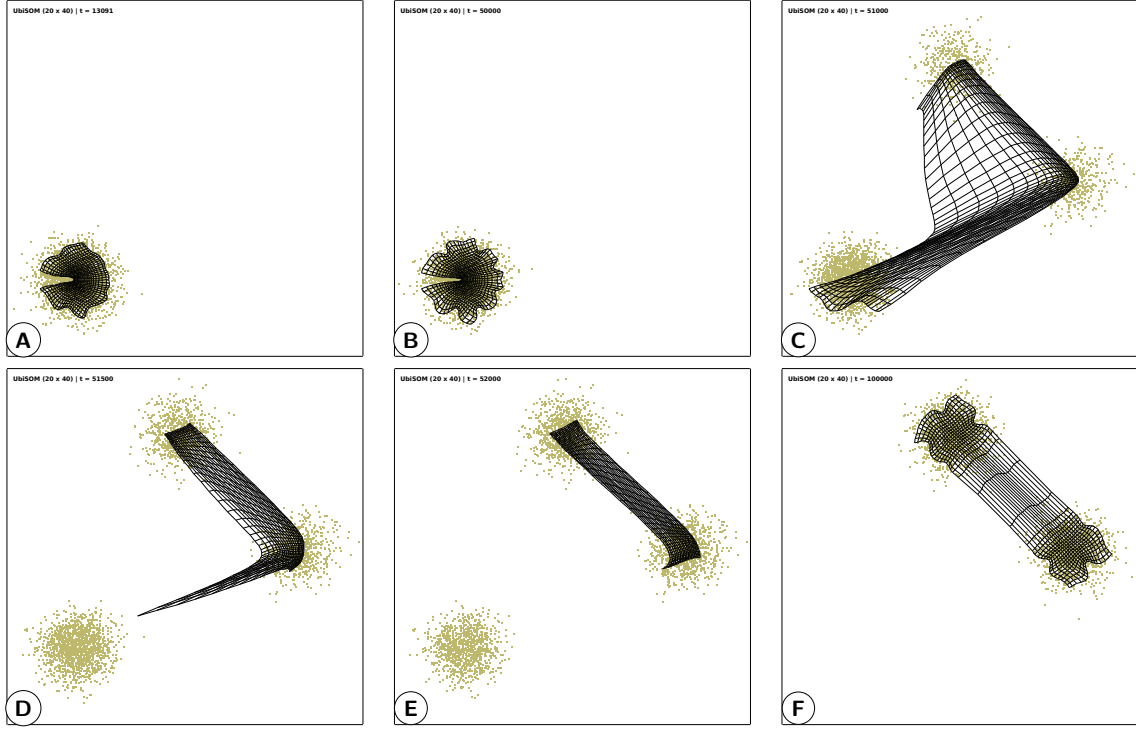
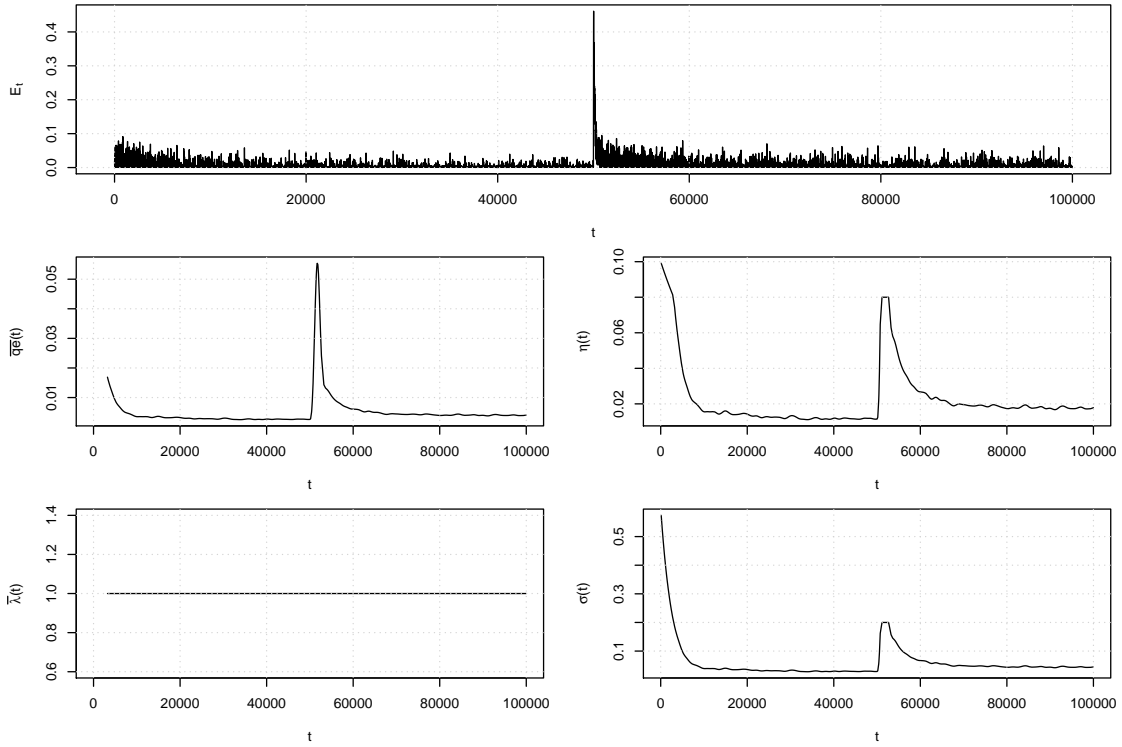


Figure C.4: The UbiSOM evolution over the stationary *Gauss* data stream, with  $T = 1500$  and  $\beta = 0.7$ .

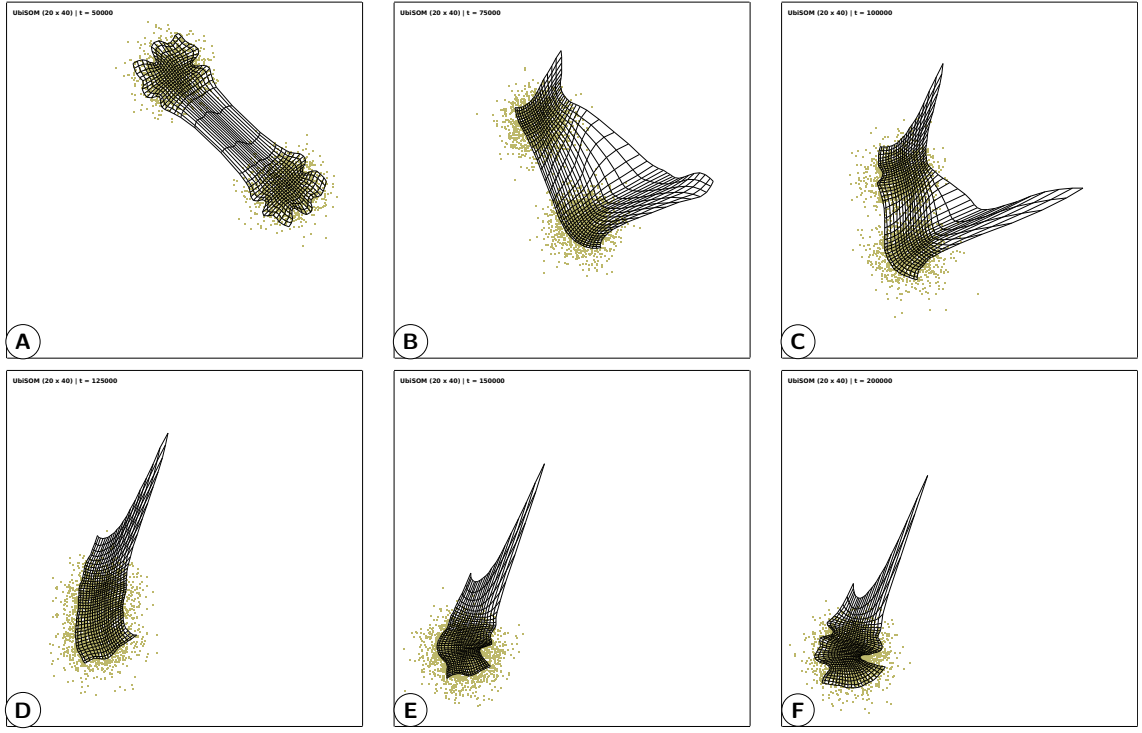


(a) Convergence of the map along time: (A)  $t = 13\,091$ ; (B)  $t = 50\,000$ ; (C)  $t = 51\,000$ ; (D)  $t = 51\,500$ ; (E)  $t = 52\,000$ ; (F)  $t = 100\,000$ .

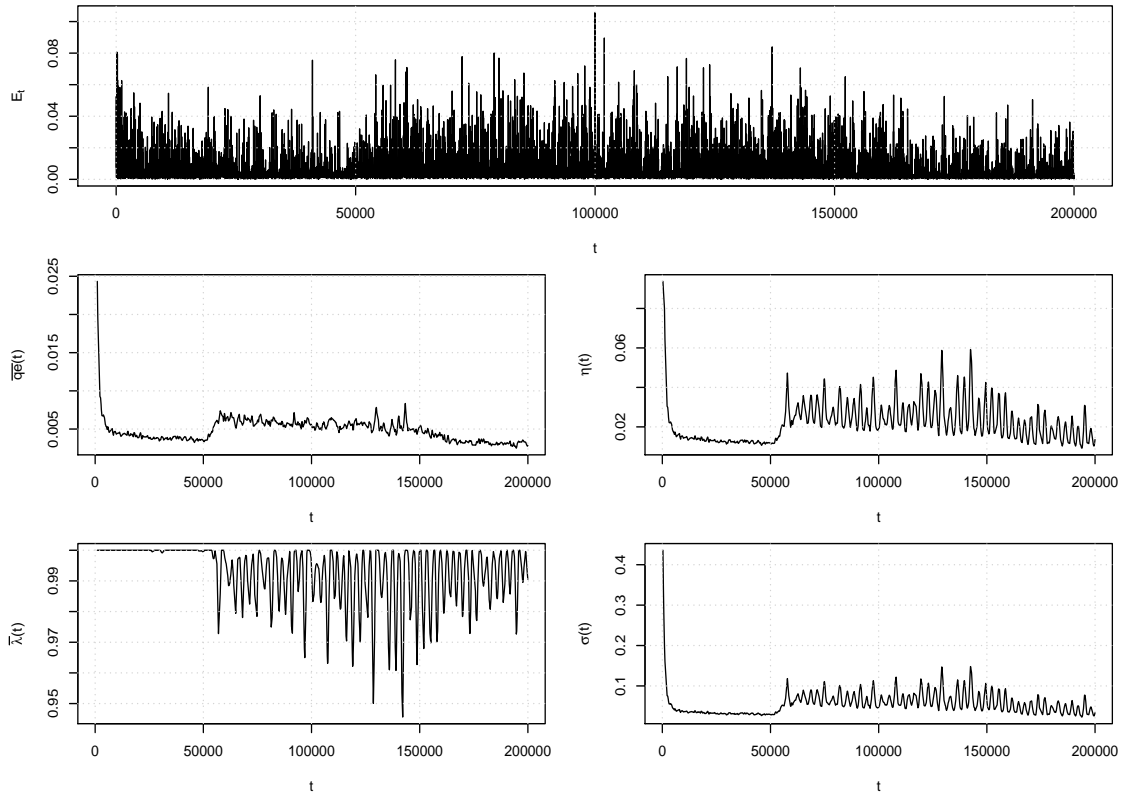


(b) Assessment metrics and learning parameters evolution.

Figure C.5: The UbiSOM evolution over the non-stationary *AbruptOneTwo* data stream, with  $T = 3000$  and  $\beta = 0.7$ .

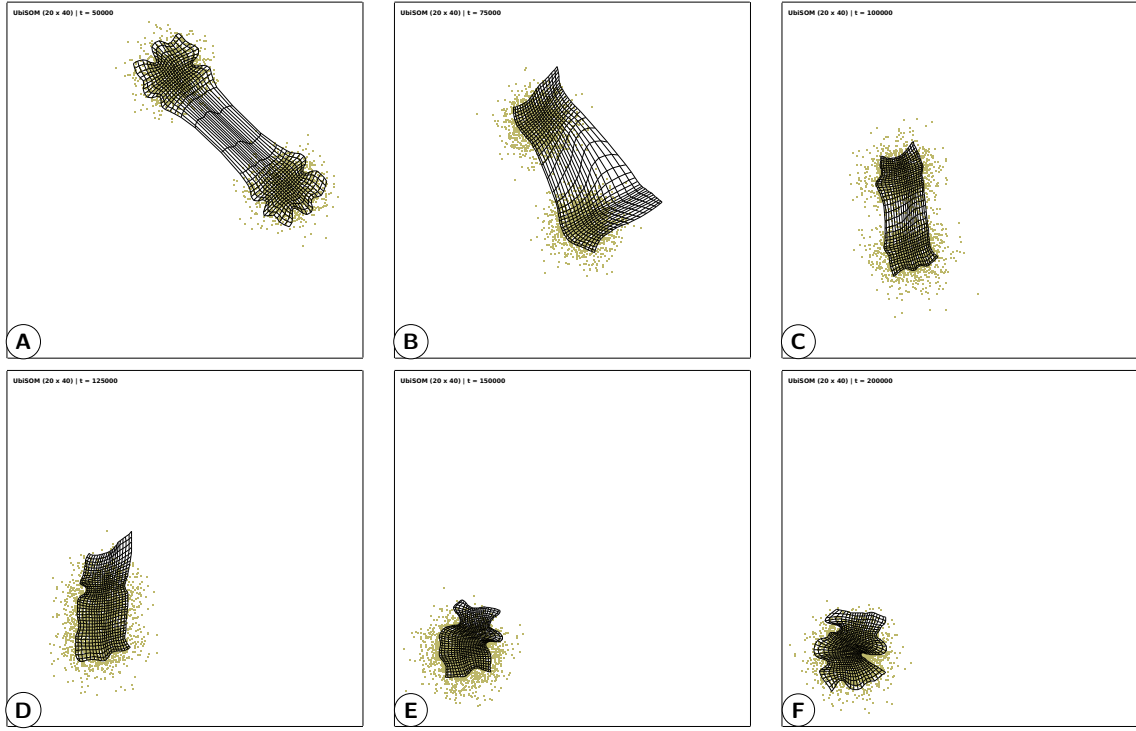


(a) Convergence of the map along time: (A)  $t = 50\,000$ ; (B)  $t = 75\,000$ ; (C)  $t = 100\,000$ ; (D)  $t = 125\,000$ ; (E)  $t = 150\,000$ ; (F)  $t = 200\,000$ .

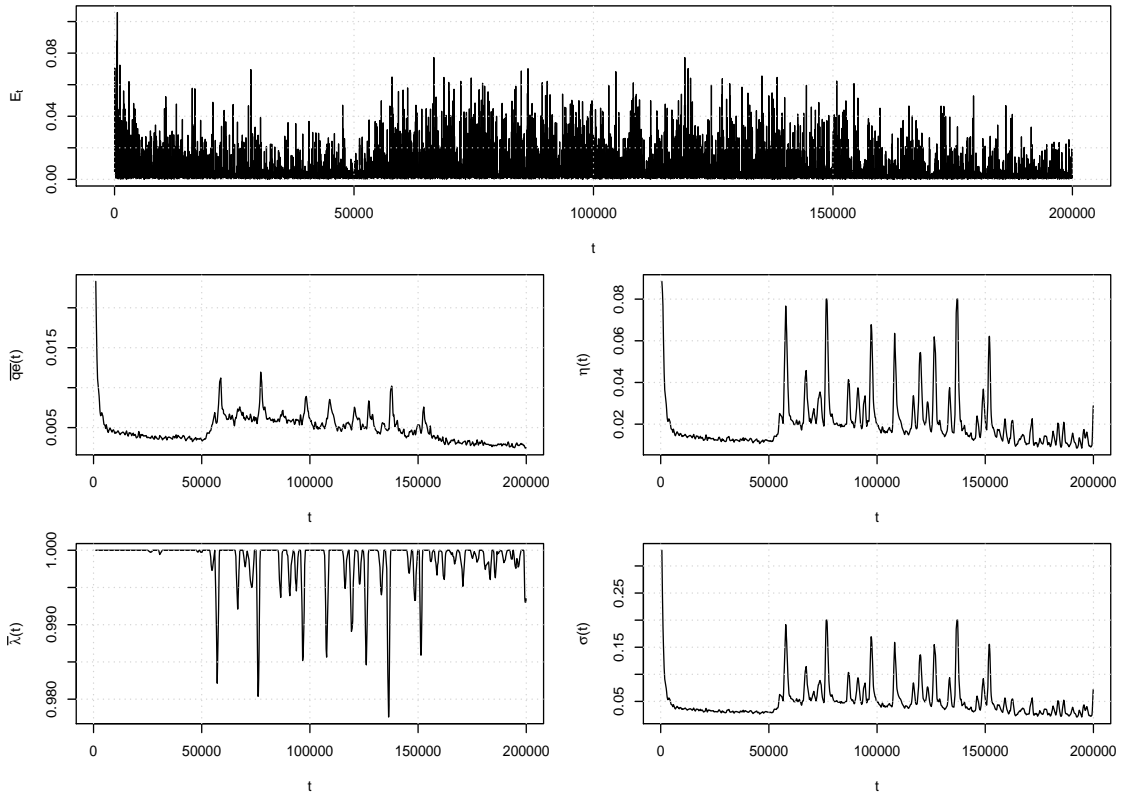


(b) Assessment metrics and learning parameters evolution.

Figure C.6: The UbiSOM evolution over the non-stationary *DriftTwoOne* data stream, with  $T = 1000$  and  $\beta = 0.8$ .



(a) Convergence of the map along time: (A)  $t = 50\,000$ ; (B)  $t = 75\,000$ ; (C)  $t = 100\,000$ ; (D)  $t = 125\,000$ ; (E)  $t = 150\,000$ ; (F)  $t = 200\,000$ .

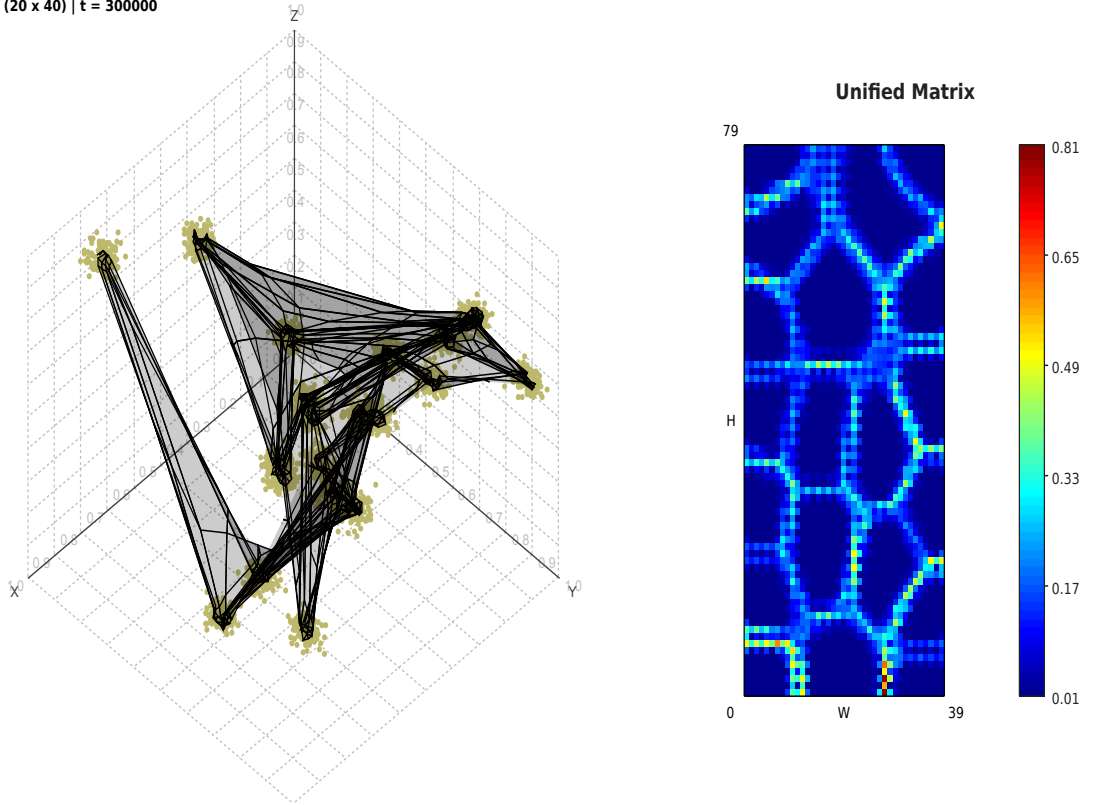


(b) Assessment metrics and learning parameters evolution.

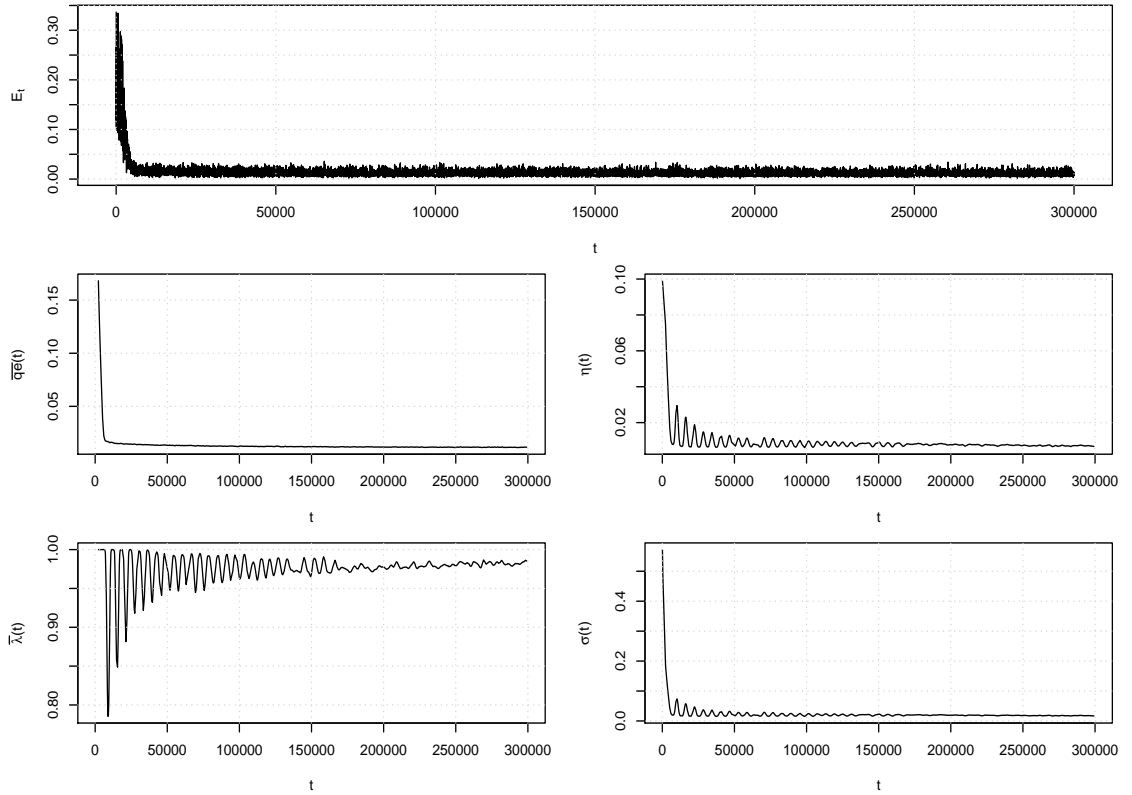
Figure C.7: The UbiSOM evolution over the non-stationary *DriftTwoOne* data stream, with  $T = 1000$  and  $\beta = 0.5$ .



UbiSOM (20 x 40) |  $t = 300000$



(a) Final map and corresponding U-Matrix. Only the first 3 dimensions were used to illustrate the map.



(b) Assessment metrics and learning parameters evolution.

Figure C.8: The UbiSOM evolution over the stationary *d5k20* data stream, final lattice and corresponding U-Matrix, with  $T = 2000$  and  $\beta = 0.7$ .





## Triple-Cascaded Moving Average

Assessment metrics in the UbiSOM algorithm, namely the *average quantization error* and *average neuron utility*, are computed over a sliding window of fixed size, e.g., a sliding window of size  $T$ . The objective is to determine if the current codebook can adequately describe the underlying distribution. Hence, error/utility values obtained during the iterations of the algorithm are averaged using a moving average. In filter theory, such average can be regarded as a linear filter, more precisely a *finite impulse response* filter. However, the output of such filters can produce undesirable results if we intend to estimate learning parameters based on this output, i.e., we are interested in monotonically decreasing/increasing values without discontinuities (smooth responses); also, we want to minimize the impact of noise (produces sporadic higher error values).

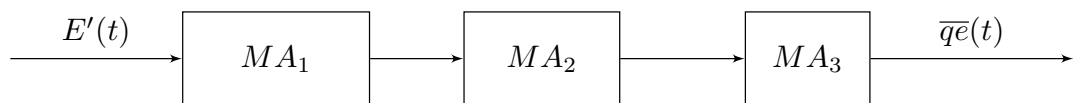


Figure D.1: Triple moving average (3MA) filter, by cascading three moving averages of decreasing window sizes.

Smoother filter responses can be obtained with *Gaussian filters*, at the expense of a higher computational cost. A Gaussian filter is considered the ideal time domain filter whose impulse response is an approximation of a Gaussian function (Blinchikoff and Zverev 2001). It also has additional advantages when compared to linear filters, namely minimizing the rise and fall time of the response. To overcome the additional computation overhead, Gaussian filters can be approximated by cascading  $n$  linear filters (Wells 1986). A “cascade” means we feed the output of the first stage (a single linear filter) into the input of the next stage and so on.

The Gaussian filter in the current implementation is approximated by three simple moving averages ( $n = 3$ ), by decreasing the respective window lengths by a specific factor (Figure D.1). These individual window sizes are decreased by a factor of 1.2067. This specific factor was derived numerically and isn't available in any scientific literature at the time of writing, but only in a web forum<sup>1</sup>. However, this factor was empirically validated during this research by obtained results, which are briefly presented in this appendix.

Comparatively, for a simple moving average (MA) of length  $T$ , the 3MA imposes the computation of the three decreasing window sizes that obey the aforementioned factor. Hence, the problem is establishing the window size  $M$  of the first stage, which is derived in Eq. (D.1).

$$\begin{aligned} \frac{M}{1.2067} + \frac{M}{1.2067^2} + \frac{M}{1.2067^3} &= T \\ M \cdot 2.8458 &= T \\ M &= \frac{T}{2.08458} \end{aligned} \quad (\text{D.1})$$

As an example, calculating  $M$  for  $T = 1000$  yields

$$M = \frac{1000}{2.08458} \approx 479.712.$$

The set of desired window sizes is derived from Eq. (D.1) as

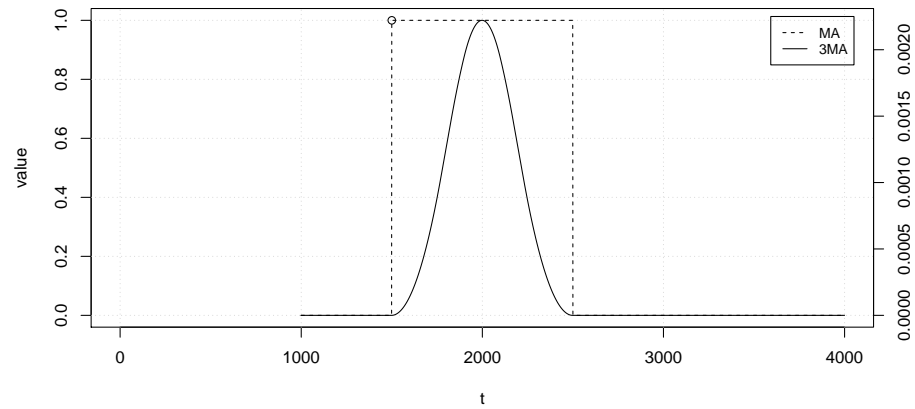
$$\left\{ \frac{479.712}{1.2067}, \frac{479.712}{1.2067^2}, \frac{479.712}{1.2067^3} \right\}.$$

The consecutive window lengths are finally obtained by rounding the previous values to their closest integers, i.e.,  $\{398, 329, 273\}$ .

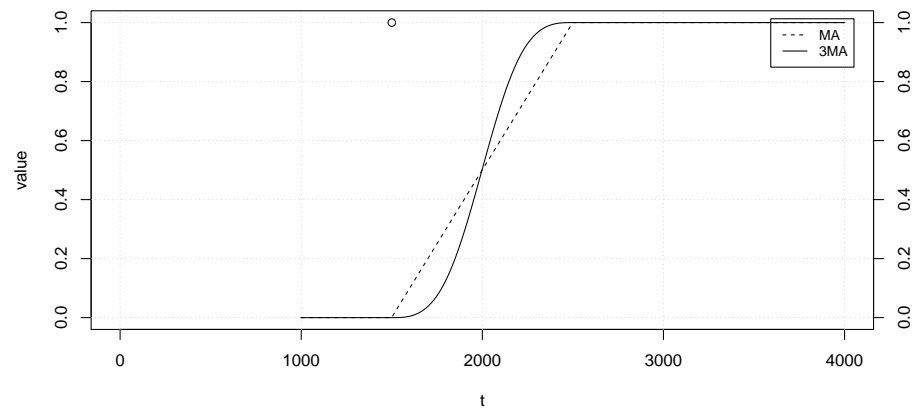
Figure D.2 depicts a comparison between MA and 3MA filters. In Figure D.2a the impulse response for each filter with  $T = 1000$  is shown, where the input is always zero with the exception in  $t = 1500$  where it is one (the impulse). In Figure D.2b the step response for the same window size is illustrated. In both figures the scale on the left pertains the input values, i.e., impulse and step, and the scale of the right pertains the output of the filters.

From these comparisons we can observe that 3MA clearly produces a smoother output with no abrupt changes in the output. If we consider the inputs as errors, then the response to noise (the impulse) has less impact in the 3MA filter, because it only reaches the highest response during a very short period of time. In case of true change (the step), the 3MA minimizes the rise time. Another important thing that can be extracted

<sup>1</sup><https://judithcurry.com/2013/11/22/data-corruption-by-running-mean-smoothers/#comment-417814>



(a) Impulse response.



(b) Step response.

Figure D.2: Comparison between MA and 3MA filters.

from these comparisons is that the Gaussian-like moving average effectively gives more importance to the later inputs. As such, the 3MA is considered ideal for the UbiSOM learning parameter estimation.

**Exploratory Cluster Analysis from Ubiquitous Data Streams using Self-Organizing Maps.**

© 2016 Bruno Miguel Nunes da Silva